Software Framework for Managing Heterogeneity in Mobile Collaborative Systems

Carlos D. Correa and Ivan Marsic
Rutgers University
ECE Department and the CAIP Center
96 Frelinghuysen Road
Piscataway, New Jersey 08854-8088 U.S.A.
+1 732 445 0542
{cdcorrea, marsic}@caip.rutgers.edu

ABSTRACT
Heterogeneity aspects in mobile collaborative systems, such as differences in user's interest, semantic conflicts across different domains and representations, and disparate device capabilities, cause difficulties in developing software applications. One of the key problems for collaborative applications is maintaining a consistent shared state.

In this paper, we describe a framework that manages several aspects of heterogeneity to maintain consistency across the collaborating sites. We assume graph data structure for application state representation. Our framework is based on structural and semantic mappings between graph structures. The mapping can be customized to meet different requirements through user-defined policies and rules. An important constraint is efficient use of scarce system resources.

We describe several applications built using the framework to collaboratively share XML documents. The XML documents in our case are 2D/3D representations of virtual worlds. We also show the performance results of our framework which demonstrate its feasibility for mobile scenarios.

Categories and Subject Descriptors
H.5.3 [**Information Interfaces and Presentation**]: Group and Organization Interfaces; H.2.5 [**Database Management**]: Heterogeneous Databases—*Data translation*; D.2.11 [**Software Engineering**]: Software Architectures

General Terms
Algorithms, Performance, Design, Experimentation

Keywords: Consistency maintenance, content adaptation, scene simplification, virtual worlds, mobile computing

1. INTRODUCTION
Computing platforms are rapidly becoming more heterogeneous, primarily in terms of device capabilities: display characteristics, CPU speeds, memory, and network connectivity. Small portable devices, such as handheld computers, smart phones and wireless watches, are becoming popular means to access the Internet. Traditional application frameworks have concentrated on systems of similar machines. However, with heterogeneous devices proliferating, there is a need for applications that are tailored to the capabilities of their current computing and communications environment. Such tailoring results in differences in data representations, GUI structures and behaviors on different platforms. There have been heterogeneous platforms and applications in the past, but what is new is ubiquitous connectivity that allows them to access and share the same data and collaborate at the same time.

Our work concerns the interoperation of these applications so that users can equitably and efficiently employ heterogeneous computer devices. Some typical application scenarios for using diverse platforms include teams collaborating on shared data, or pervasive computing, where the application state is available anywhere and the user may at different times employ unlike devices to access and edit the same document.

The application state consists of all application's data structures that are relevant for the process of sharing. An example of application state is what is usually called scene graph in two-dimensional and three-dimensional graphics applications. It is a hierarchical description of scene objects, with part-whole representation at different levels of abstraction. Other examples include conceptual knowledge representation using relationships such as inheritance, spatial or temporal proximity, etc. There are many different data structures, but in this work we focus on the graph data structure as a canonical representation, since it is the most common one and the results here can be relatively easily generalized to other data structures.

A key issue in interactive information sharing is maintaining consistent state across the applications running at distributed sites. This is required in order to have meaningful collaboration; otherwise the application states at different sites would diverge, in which case they could as well be considered as entirely disconnected from each other. The term "consistency" is used in different contexts. In this paper, consistency means the absence of logical contradictions between the replicated copies of a data object. Maintaining consistent state of replicated clients in homogeneous systems is a well-researched problem, see e.g., [1][20], and is not addressed here. However, in addition to the problems inherited from homogeneous collaborative applications, there are some problems

GROUP'03, November 9–12, 2003, Sanibel Island, Florida, USA.

specific to applications with heterogeneous representations and the latter are the focus of this paper.

In this paper, we present a framework that manages several aspects of heterogeneity in collaborative systems: differences in user's interest, semantic conflicts across different domains and representations, and disparate device capabilities. The key objective is maintaining consistent application states between the participants. The framework, described in Section 3, is based on graph mappings and uses topological transformations of the data structures that can be customized through semantic mappings and user-defined policies. Section 4 describes several applications of the framework, each of them aimed to solve a particular issue in heterogeneous systems. In Section 5 we describe the results obtained in the different applications, in terms of performance. Finally, we present conclusions in Section 6.

2. RELATED WORK

To our best knowledge, there has been no other work on consistency in collaborative applications with many-to-one (lossy) mappings between the data representations and with bi-directional interactions. Consistency maintenance is a traditional problem in databases [6]. Databases are mostly concerned with the *process* view of consistency, in the sense of ordering and closure constraints on the propagation of write operations. Our focus here is on the *semantic* view of consistency, in the sense that different representations of meaning of database elements are not logically contradictory. Work on interoperating databases, e.g., [2][4][19], concerns itself with mapping queries so the data can be accessed from heterogeneous databases. However, it assumes that all domains have exact (i.e., one-to-one) correspondence, and since our main focus is on representations with many-to-one correspondence, this work is of limited applicability here. There has been work in supporting transparent collaboration between heterogeneous applications, particularly single-user editors [11]. However, they also assume a one-to-one mapping of operations (editor primitives). Our approach extends to the cases where many-to-one mappings are needed.

In addition to ensuring consistency, the framework presented in this paper provides a mechanism to reduce the resource demand. Simplifying or "compressing" structured content is a common practice, analogous to signal or image compression. Lossy simplification of structured content ranges from reducing the number of polygons of a 3D representation [9][17], through MPEG-4 video compression [18], transcoding of Web content [3][8][12], to text summarization [13][14]. However, these all address *one-way* content compression.

We believe that there will arise the need for collaboration and *bi-directional* exchange on documents compressed to different degrees at different collaborating sites. This paper addresses the related issues.

A key requirement in mobile systems is that the support for disconnected operation. Wireless links often provide only intermittent connectivity, so it is desirable to maintain a local state to enable the user to continue the work while disconnected, and to re-synchronize when the client gets opportunity to reconnect. Some initial results already exist [16][21], and our team is currently addressing this issue as well.

3. CONSISTENCY MAINTENANCE FRAMEWORK

Our architecture is based on the assumption that collaborative systems represent the shared information as *graph data structures*. As long as each collaborating application presents a graph-based API to its state, the application can actually represent the application state using some other data structure. We also assume that the shared information is customized in a possibly *lossy* manner to each collaborating site. We first consider the client-server case and then generalize it to the peer-to-peer case.

Without loss of generality, let us consider the case of a single server and a single client. The framework easily extends to the case of multiple clients and servers. We assume that the server maintains the complete application state as a graph G_s. Because G_s contains all the information, we call it *super-graph*. The application state on the client, denoted as a graph G_c, is typically less detailed, and it never contains more information than G_s.

Collaboration is enabled by defining a mapping between G_c and G_s, and ensuring that operations on G_c are consistently mapped onto operations on G_s and vice versa. The concept of consistency is defined below.

3.1 Lossy Graph Mappings

The mapping between server and client graphs is obtained by means of three types of mappings:

1. Topology mappings
2. Property mappings
3. Policies (both application-dependent and user-defined) for resolving mapping ambiguities

The above mappings are increasingly more specific to the individual requirements of the clients. This means that while topological mappings can be used in the great majority of collaborative systems, property mappings have a more restricted applicability in certain domains, and policies are even more specific to the application needs and the user interests.

3.1.1 Topology Mappings

Although there are many ways to define lossy mappings between the server and client graphs, most of the applications make use of three types of mappings, when simplifying the application state in order to address the heterogeneity issues: *subgraph mapping*, *vertex contraction* and *path merging* (see Figure 1).

For a given graph G, let $V(G)$ denote the vertex set, $E(G)$ the edge set, and $P(G)$ the path set (including the edges, which are paths of length 1).

Definition 1. *Subgraph mapping is an isomorphism from a subgraph G_s^* of G_s to G_c.*

This is perhaps the simplest mapping between the server and client graphs and is the basis of filtering out irrelevant information, as depicted in Figure 1(a).

Definition 2. *Vertex contraction is an isomorphism from G_s' to G_c where G_s' is obtained by contracting a subset of vertices in G_s to a single vertex.*

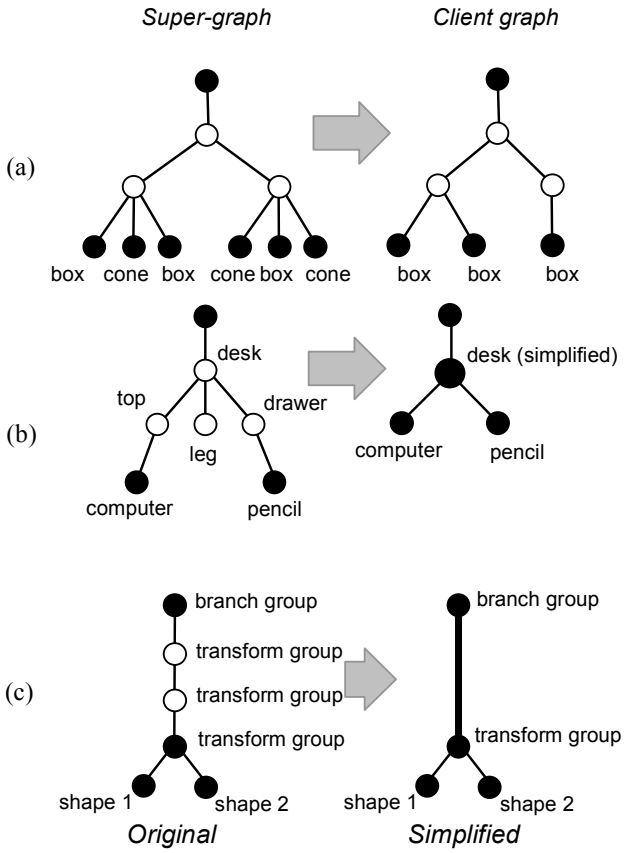

Figure 1. Examples of lossy topology mappings: (a) Subgraph mapping; (b) Vertex contraction; (c) Path merging.

This is perhaps the most common type of mapping in shared virtual environments, which allows the simplification and abstraction of structured data as depicted in Figure 1(b).

Definition 3. *Path merging is an isomorphism from G_s' to G_c where G_s' is obtained by merging a path in G_s, where all internal vertices have degree 2, to an edge*.

This type of mapping allows the omission of hierarchical relations between the elements, useful for flattening hierarchies, as depicted in Figure 1(c).

3.1.2 Property Mappings

Since it is not possible to enable a complete interoperation only by means of topology mappings, the framework also defines a set of property mappings. The property mapping is related to the topology mapping defined above as follows. In the case of subgraph mapping, the property conversion is usually a one-to-one function. In the case of vertex contraction, the properties are usually defined as a combination of the properties of the grouped vertices. In addition to this, the framework also considers the cases of property dependency, as described below.

The general sever-client mapping is defined as follows: a server property p, defined as a tuple (*type*, *value*) is mapped to a client property q, such that:

$$q = M_p(p)$$
$$q.value = f\left(\bigcup_{w \in W} properties(w)\right) \quad (1)$$

for $W \subseteq V(G_s)$ and $p \in properties(v)$, for some $v \in W$

where M_p is a one-to-one mapping of properties. Eq. (1) provides a generic way to express property dependencies. Inverse mapping of a client property q is computed as follows:

if $f^{-1}(q)$ exists, then $p.value = f^{-1}(q)$

otherwise, it is defined by a user-defined policy

An example of this mapping is coordinate conversion between 2D and 3D virtual environments. The property *position* in the 3D domain, consists of a tuple (x,y,z). Conversion to the 2D domain is done through a simple function:

$Position_{2D}.value = (x,y)$, where $Position_{3D}.value = (x,y,z)$

However, it is not possible to compute the inverse function deterministically. If the 2D application creates a vertex with property p = (*Position*, (x,y)), this applies to infinite positions in the 3D environment along the z-axis. A possibility is to use a resolving policy, so that inverse mapping can be determined. For example:

$Position_{3D}.value = (x,y,0)$, where $Position_{2D}.value = (x,y)$

Property mapping for 3D to 2D coordinates can be more complex, since they need to handle rotations and scales in addition to positions, and they may depend on the coordinate systems of the ancestors in the graph structure.

3.1.3 Application-dependent and User-defined Policies

Since the server-client mapping is not a one-to-one function, in some cases it is not possible to determine the inverse function for a given client operation. User-defined policies fill the gaps where topology and property mappings are unable to make such a decision.

Let us consider the following case: a group of server vertices v_1, v_2, … , v_n is contracted into a client vertex w. When the client adds an edge from w to another vertex w', it may be necessary to add several edges from v_1, v_2, … , v_n to some other vertices at the server. The exact number of new edges generated at the server is application-dependent and cannot be determined by the topology mapping alone. In such case, an application-dependant or a user-defined policy will determine the correct operation.

3.2 Mapping Specification

On of the greatest challenges in this framework is how to define the initial mapping for the different elements in the collaborative system.

We consider two different ways to obtain the initial mapping: *a rule-based mechanism* and an *automated specification*.

3.2.1 Rule-based Specification

The rule-based mechanism is composed of a rule set $R = \{r_1, \ldots r_n\}$, where r_i is a tuple of the form (p, a), where p is a predicate expression and a is a topological action.

Whenever an element is created, it is validated against the rules in

```
Set LOD=3            // level-of-detail
Rule 1:  v.depth<LOD    // tree depth
         Action:  subgraph.insert(v)
Rule 2:  v.depth=LOD
         Action:  createSet Vs
                  Vs.insert(v)
                  vsets.insert(Vs)
Rule 3:  v.depth>LOD AND ∃s ∈ vsets: v.parent ∈ s
         Action:  s.insert(v)
```

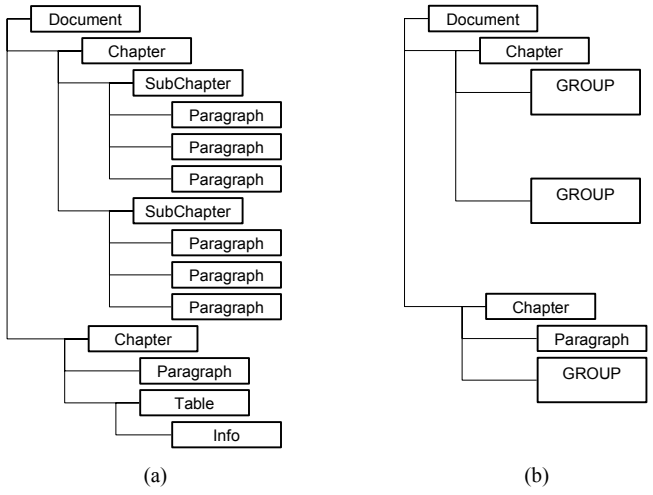

Figure 2. Example of a rule-based mapping specification. (a) Super-graph; (b) Resulting client graph.

order to determine the appropriate mapping, i.e., whether is mapped into a single vertex of the client graph, is contracted with other vertices in a grouping vertex, or not mapped at all.

Figure 2 shows an example of a set of rules and their corresponding actions. In this example, the mapping is used to simplify a tree structure in a way such that nothing below a given tree depth is sent to the client.

3.2.2 Automated Specification

The automated mapping specification, unlike the rule-based one, is performed over a set rather than on individual elements. One of its applications is constraint-based mapping. In this application, the mapping is defined in such a way that some constrained variable, e.g., resources, is kept below a given threshold. This is useful for supporting those clients with limited resources, such as bandwidth, memory and computing power. In Section 4.3 below we describe an application that uses this kind of automated mapping, which not only satisfies the given constraints but also maximizes the user's utility.

3.3 Maintaining Consistency

In order to provide a meaningful collaboration, the different clients must be able to interoperate consistently. Each of them will have a different application state, according to the particular mapping specification.

In this context, we define consistency as follows. First, we require that every pair (G_s, G_c) generated by the techniques in Section 3.2 satisfy the following conditions:

(A1) There is a subset V^* of the vertex set $V(G_s)$, such that there is a surjective mapping $M_v : V^* \rightarrow V(G_c)$.

(A2) There is a subset P^* of the path set $P(G_s)$, such that there is a surjective mapping $M_p : P^* \rightarrow E(G_c)$. If v_1 and v_2 are the end vertices of a path $p \in P^*$ then $M_v(v_1)$ and $M_v(v_2)$ are the end vertices of the edge $M_p(p)$.

(A3) For each path $p \in P^*$, the internal vertices of p are not in V^*. Paths in P^* do not intersect and no two paths in P^* share the same pair of end vertices.

Definition 4. We say G_c is *consistent* with G_s with respect to (M_v, M_p) if M_v is a surjective mapping from V^* to $V(G_c)$ and M_p is a surjective mapping from P^* to $E(G_c)$ for some $V^* \subseteq V(G_s)$ and $P^* \subseteq P(G_s)$ such that conditions (A1), (A2) and (A3) hold.

The operations that can be applied to the application state (graph) at any of the collaborating sites are as follows:

1. Topological operations: add/remove a vertex and add/remove an edge
2. Property operations: add/remove/modify a property, where a property is a tuple of the form (name, value)
3. Application- and User-defined operations: combinations of the above, e.g., copy/paste a subgraph can be defined in terms of vertex and edge additions

In order to guarantee consistency, we have defined several algorithms that map single operations on G_s into operations on G_c and vice versa. Here we present a brief summary, and the details are available in [15].

3.3.1 Case 1: Consistency for Subgraph Mapping

For the case of a client's graph being isomorphic with a subgraph of the super-graph (Definition 1) maintaining consistency is straightforward: if the operation on the super-graph refers to an element (vertex or edge) with a direct mapping in the client graph, then the operation is also performed on the corresponding vertex in the client graph.

3.3.2 Case 2: Consistency for Vertex Contraction

In the case of the client's graph obtained by contracting a subset of vertices in G_s (Definition 2), the algorithm works as follows:

(1) When adding vertex v to G_s: Either $V_s(t+1) = V_s \cup \{v\}$ or nothing happens, depending on the mapping specification

(2) When removing a vertex v from G_s:

If $v \notin V_s$ Then it is treated as in Case 1

Else $G_c(t+1) = G_c(t)$, $V_s(t+1) = V_s(t) / \{v\}$

This assumes that $|V_s| > 1$, i.e., a subset of vertices always contains more than one vertex. In the case where there is only one vertex in subset V_s (by subsequent deletions), this is in fact a mapping of type 1, so it is not longer consider a vertex set.

(3) Adding an edge (uv) to G_s:

If $u,v \in V_s$ then $G_c(t+1) = G_c(t)$

Else If $u \in V_s$, $v \notin V_s$ then $G_c(t+1) = G_c(t) \cup (M_v(v)v_s)$

Else $u,v \notin V_s$, Considered in Case 1

(4) Deleting an edge (uv) from G_s:

If $u,v \in V_s$, then $G_c(t+1) = G_c(t)$

Else If $u \in V_s$, $v \notin V_s$ then $G_c(t+1) = G_c(t) \cup (M_v(v)v_s)$

Else $u,v \notin V_s$, Considered in Case 1

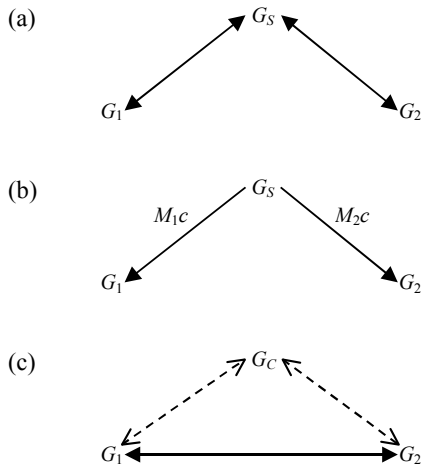

Figure 3. Setting up a peer-to-peer communication between two clients. (a) Initial state download; (b) Mappings $M_{<i>}c$ distributed for peer-to-peer operation; (c) Peer-to-peer operation.

For the case of the inverse mapping, i.e., from client operations to server operations, similar algorithms are defined. However, because it is a many-to-one mapping, there is no deterministic way to obtain an inverse. In such case, we enhance the topology mapping with application-dependent and user-defined policies, which decide the appropriate action in those cases where the inverse mapping is not possible to be computed.

3.3.3 Case 3: Consistency for Path Merging

Since the applications described in this paper only exploit the first two cases of mapping, path-merging algorithms are not considered. See [15] for details on path-merging algorithms.

3.4 Peer-to-Peer Collaboration

An interesting extension of this framework is peer-to-peer collaboration, where the peers have limited resources and each stores only part of the shared information. Let us consider the following scenario: two clients with graphs G_1 and G_2 are connected to a server with graph G_s. At some point, they may be disconnected from the server and interact with each other directly.

One way to solve this in the general case is to allow one of the clients to obtain the mapping of the other client and let it act as a server. This solution, however, is just a variation of the client-server architecture. For more than two clients, it is possible to create a hierarchy of clients such that some of them act as servers to others. This can be easily built from the basic architecture, but it is no longer a peer-to-peer scenario.

Another way to interoperate the two clients is for each client to compute the intersection of their graph mappings. With this additional information, it is possible to determine whether an operation in G_1 must be mapped to client G_2. The problem with this approach is that this extra information can be potentially large, and may result in making both client graphs homogeneous with the super-graph.

We can still solve the peer-to-peer collaboration problem by restricting the number of intersections of the graph mappings. Instead of constraints (A1), (A2) and (A3) defined in Section 3.3, we impose new constraints as follows:

(B1) The client graph is defined by a subset of vertices V^* and a partition Π of V^*. All vertices in the same member of Π, which is a subset of V^*, are mapped to the same vertex in the client graph.

(B2) The mapping of the edges is induced by the mapping of vertices.

(B3) Let (V_1, Π_1) and (V_2, Π_2) be subsets and partitions that define two client graphs. Let U_1 be a member of Π_1 and U_2 a member of Π_2. Then either $U_1 \cap V_2$ and $U_2 \cap V_1$ are disjoint or one is a subset of the other.

These constraints are reasonable in a wide range of applications that use techniques such as summarization, abstraction or simplification of hierarchical structures, to adapt to the limited system resources.

The scenario for setting up a peer-to-peer communication is depicted in Figure 3 and works as follows. Initially, two clients are connected to a shared server at least for the time to download the documents to collaborate on, Figure 3(a). When both clients are interested in setting a peer-to-peer connection, the mapping intersection is computed at the server site such that a new graph G_C (subgraph of G_s) acts as a super-graph for G_1 and G_2. The mappings to this new graph (M_1c and M_2c) are sent to the respective clients, Figure 3(b). After receiving the mappings, both G_1 and G_2 can communicate directly as if they were interacting with a virtual server G_C, Figure 3(c). Constraints (B1), (B2) and (B3) ensure that the additional information sent as the mappings $M_{<i>}c$ to the clients is relatively small.

3.5 Intermittent Connectivity and State Merging

As mentioned earlier, it can be expected that the user will often operate in disconnected mode since wireless connectivity is unreliable and unpredictable. Upon reconnection, the application state needs to be merged with the rest of the collaborative system. This issue is present in homogeneous systems as well, and we can use the same techniques in the case of heterogeneous representations. Some recent systems are specifically targeted for state merging in mobile applications [16][21], but they assume homogeneous application states.

The latecomer support is related to state merging. It is interesting to notice that the same mapping specification for the same computing platform may produce different graphs on different clients. For example, an automated mapping at time t produces a graph $G_1(t)$ for Client 1. As the users interact with the system, the graph at $t+N$ will be $G_1(t+N)$. If a Client 2 with the same characteristics as Client 1 joins the session at $t+N$, the automated mapping will produce for Client 2 $G_2(t+N) \neq G_1(t+N)$. This is since the automated mapping tries to fit the maximum number of graph elements to the available resources, whereas mappings individual graph operations follow different rules. However, this difference presents no problem for the algorithm's correct functioning.

3.6 Strengths and Limitations

The framework can be applied to the great majority of the collaborative systems, where the graph is a common data structure. We provide a generic solution to the interoperation of disparate applications by dividing the mapping in the three types described in Section 3.1. The basic requirement for our framework is the use of a

```
<DEFINE name="types"
type="consistency.graph.SimpleSetImpl">
   <ELEMENT>Document</ELEMENT>
   <ELEMENT>Table</ELEMENT>
   <ELEMENT>Paragraph</ELEMENT>
   <ELEMENT>Title</ELEMENT>
   <ELEMENT>Subtitle</ELEMENT>
</DEFINE>
<RULE>
   <BIND>types</BIND>
   <EXISTS object="x" set="types">
      <EXPRESSION object="v" property="type"
      operator="equals" value="x"/>
   </EXISTS>
   <ACTION>
      <SETACTION object="v" set="subgraph"
      operator="insert"/>
   </ACTION>
</RULE>
```

Super-Graph:

```
<DOCUMENT name="myDoc">
   <TITLE>My Title</TITLE>
   <PARAGRAPH>This is a paragraph
      <IMAGE src="source" />
      <IMAGE src="source2" />
   </PARAGRAPH>
   <SUBTITLE>Subtitle</SUBTITLE>
   <PARAGRAPH>Another paragraph
      <FOOTNOTE>Include a footnote here
      </FOOTNOTE>
   </PARAGRAPH>
</DOCUMENT>
```

Client Graph:

```
<DOCUMENT name="myDoc">
   <TITLE>My Title</TITLE>
   <PARAGRAPH>This is a paragraph</PARAGRAPH>
   <SUBTITLE>Subtitle</SUBTITLE>
   <PARAGRAPH>Another paragraph</PARAGRAPH>
</DOCUMENT>
```

Figure 4. XML definition of user interests and example graphs.

directed graph as the underlying data structure for application state representation.

A key feature of the framework is that it provides consistent application states between the participants of the collaborative system. Here, consistency means the absence of logical contradictions between the replicated copies of a data object. Consistency is maintained through the preservation of the graph mapping conditions for any given graph operation. It is important to note that this concept of consistency is different from data integrity under concurrent updates [1][20], which is a necessary requirement of most collaborative systems. Our framework can and should coexist with traditional concurrency control mechanisms, since we do not impose a particular method for implementing graph operations.

In addition, the framework can be used to meet the requirements of a particular system, with the use of property mappers and policies, and the combination of such elements can be used to solve different challenges in heterogeneous systems. Section 4 below describes example scenarios of how the framework could be used in collaborative applications.

One of the limitations of our framework is the key assumption regarding the nature of the underlying data structure. For collaborative environments that share structured data, the graph data structure may not be the most efficient data type for all purposes, and some applications may suffer performance penalty due to fixing the shared document structure. For example, a spreadsheet can be more efficiently represented as a multidimensional array. The performance of a graph-based spreadsheet may degenerate for a large document. However, we believe that such cases would appear relatively rarely in practice and the gains from having a general solution far outweigh the drawbacks. Also, as already pointed out, the application may use any data structure as long as it provides a graph-based API for application state access. In addition, it appears that most future applications will exchange structured data as XML documents, which are trees—a subtype of a graph.

For the applications that work with or include non-structured elements, such as video and sound, one alternative is to use a meta-representation that can be defined in a graph. For instance, a hierarchical description of a video that reflects the internal structure of scenes, shots and frames. One example of this is MPEG-4 data representation [18].

Another limitation is the requirement of the super-graph as a central structure, which implies client-server architecture. Section 3.4 above discusses the peer-to-peer architecture, where each client has a different application state and no client has all the information as to be defined as super-graph. We found that it is not possible to solve efficiently the case of peer-to-peer communication in the case of many-to-one mappings for the generic model. However, this limitation is not specific to graph representations: It is inherent to any heterogeneous collaboration with lossy mappings between the states, regardless of the data structure used for application state representation.

4. EXAMPLE APPLICATION SCENARIOS

We have used the framework in several example scenarios, each of them exploiting different aspects of the architecture. The most significant are described in the following sections.

4.1 Interest Management and Filtering

One dimension of heterogeneity is defined by the user's interest. Collaborative users may have a different assessment of the information usefulness, depending on the context, user's role, personal preferences, etc., so they may want to subscribe to different types of information.

In our framework, user interest is defined as rules. Figure 4 shows the rule specification for interest subscription based on the data type. In this particular example, the user is interested in those objects that match the types in the set {Document, Table, Paragraph, Title, Subtitle}. The system propagates only the updates related to the objects of interest, and all other updates do not reach the user.

4.2 Semantic Mapping for Cross-domain Applications

Another dimension of heterogeneity is the semantic differences between the information representations. These may arise due to disparate applications and computing platforms or because we are interoperating existing applications with different state representations. Our framework solves this kind of problems through property mappings and dependency specifications. We have employed the framework to interoperate 3D and 2D collaborative virtual worlds, where the users can manipulate and edit the objects. In a typical scenario, a participant using the 2D application only sees a planar view of a 3D model. Although the user perceives the

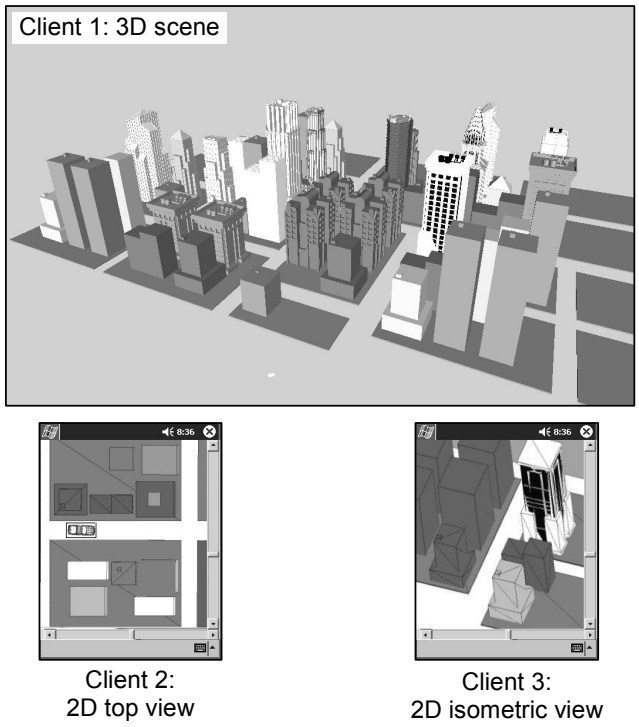

Figure 5. Example of 3D and 2D interoperation. Client 1 is a desktop PC, whereas clients 2 and 3 are PDAs.

resulting images as flattened 3D models, the application itself is only aware of 2D primitives.

An example of the application is shown in Figure 5, where a high-end client's graph (Client 1) exactly corresponds to the server super-graph. Note that it is possible to display an isometric view of the model in the 2D application. This is accomplished with a slight modification of the property mappers, namely those that convert 3D to 2D coordinates. This conversion is usually performed by multiplying the elements in 3D by a projection matrix. Depending on the point of view, the projection matrix will change.

4.3 Constraint-based Simplification of Hierarchical Data

Another aspect of heterogeneity is the differences that arise from lossy simplification of information that "compresses" the information to fit it into the scarce system resources of the client device. Availability of resources such as bandwidth, memory, and computing power, limits the amount of information that can be sent and displayed in a given device. This requires an automated specification of the graph mapping capable of meeting the given constraints.

One particular application of this scenario is the dynamic simplification of hierarchical data. Here, each vertex in the graph is assigned benefit and cost values, which represent its contribution to user's utility and resource requirements, respectively. An example of user's utility is perceptual fidelity of the scene. The graph mapping is obtained by solving a constrained optimization problem, such that the resulting client graph maximizes the benefit metric while keeping the resources below a given threshold (see example in Figure 6).

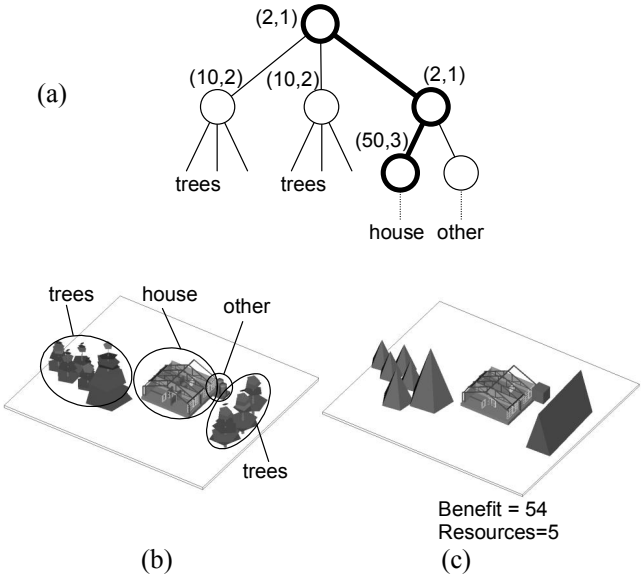

Figure 6. Example of optimal simplification (a) Scene graph with (benefit, resource) pairs; (b) Complete scene; (c) Simplified scene for max resources = 5.

In a related work [7], we developed an algorithm that solves exactly this optimization problem, which is a variation of the Tree Knapsack Problem [5]. We used the results to improve interactivity in collaborative virtual worlds on mobile devices: we achieve interactivity by guaranteeing a minimum update rate.

5. EVALUATION

We tested different collaborative scenarios for the different aspects of heterogeneity we intend to solve. The software architecture is depicted in Figure 7. In a typical scenario, 3D and 2D users collaboratively visualize a set of virtual objects, e.g., Figure 5. When a user interacts with the environment, a set of operations is performed on the application's graph. When these operations are sent to the server, they are mapped to the super-graph, according to the client's graph mapping and the algorithms described in Section 3. The operations in the server-graph are accordingly mapped into operations on the client graphs. Note that in some cases, the mapping will be specified by the set of rules, whereas in other cases it will be specified by the automated mapper (Section 3.2). In our particular application, it is the user who determines which mapping method is used. The automatic mapping simplifies a hierarchical document in the best way possible, according to some pre-defined benefit metric, such that the constraints in resources are met [7].

An important quality metric for this framework is performance. In (mobile) collaborative systems, the performance metrics critical for providing interactivity are efficient use of resources and response latency. Unfortunately, these two variables represent a tradeoff. If the application needs to meet a certain constraint in resources, particularly in wireless devices, it is necessary to run complex computations that increase the latency. Conversely, if the application needs to minimize the latency, it may have to exceed the threshold of allowed resource usage.

5.1 Resource Utilization

The tradeoff between efficient use of resources and response latency is evident in Figure 8. In this scenario, a client using a mobile device

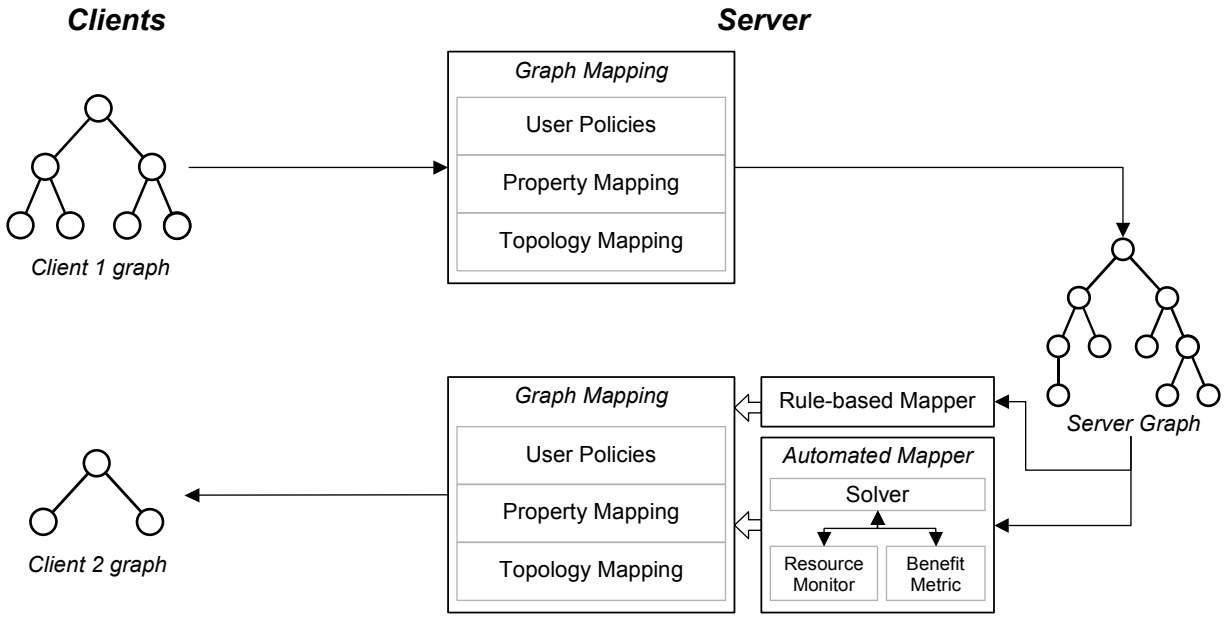

Figure 7. Software architecture of a collaborative scenario described in the text. ("Solver" is the constraint optimization solver.)

receives simplified versions of different documents. One way to achieve this is to specify a set of rules as those in Figure 2, which groups those parts of the document below a certain tree depth into single vertices. Although this method reduces the amount of information sent to the client and runs fast, it does not guarantee the optimal use of allocated resources. A way to provide such guarantee is to select the best combination of elements from the document such that resources are used to the maximum possible amount, without exceeding a given threshold. Figure 8 shows the amount of resources used for different documents (virtual objects), for both methods.

5.2 Latency

In order to measure latency, we performed both analytical and empirical studies. Latency, as perceived by the user, is the sum of several delays in the process of mapping operations from the server to the client.

$$Latency = network\_delay + t_G + max\ \{\ t_R, t_A\ \}$$

where t_G is the time to perform the graph mapping, t_R is the time to validate the rules, and t_A is the time to run the automated mapper.

Since a particular application and the user may customize the framework (by specifying property mappings and user policies) in order to meet a particular requirement, it is impossible to determine an upper bound for the server latency. For example, a property mapping might be as simple as a value conversion, which takes $O(1)$ time, or it might involve a large dependency tree, which could increase latency by an order of magnitude.

However, we have been able to bound the latency for topology mapping and t_A. Topology mappings takes an amortized time of $O(1)$. The amortized time means that, in the average, the time for each graph operation is constant. It can be seen that all the operations in Section 3.3 involve a single operation on the graph. However, some operations on vertex sets take $O(|V_s|)$ time, where $|V_s|$ is the cardinality of vertex set V_s. Since V_s was necessarily created using $|V_s|$ operations of time $O(1)$ at some point in the past, on the average each operation takes $O(1)$ time.

In practical applications, we found that the critical factor in server latency is t_A, the time required to perform automatic mapping. This is due to the fact that this automated mapper performs an optimization problem, which is known to be NP-Complete [10]. Although there are greedy algorithms that approximate the problem efficiently, there is no guarantee regarding the optimality of the solution [10]. For that reason, we have developed an algorithm that simplifies optimally a hierarchical structure in $O(nR)$ time, where n is the size of the tree, and R is the upper bound in resources (available bandwidth, memory, etc.) [7]. We have found that it is possible to reduce the computation time (server latency) in dynamic scenarios when the updates in the tree structure have locality (spatial coherence). Figure 9 graphs the computation time t_A for a user in a mobile device navigating through the shared world from Figure 5. The mobile device used is a Compaq iPAQ Pocket PC H3870. In this scenario, the client on the mobile device is not capable of showing the entire virtual world. Instead, a simplified version is retrieved from the server, such that the elements closer to the user's focus of attention are represented in more detail than the rest.

5.2.1 Adaptation to Latency

In addition to server latency due to graph mapping, network delay is substantial in wireless connections. For that reason, and the fact that delay may vary arbitrarily for different operations, it is necessary to adapt to latency. We perform the adaptation by setting a maximum update rate, computed as follows.

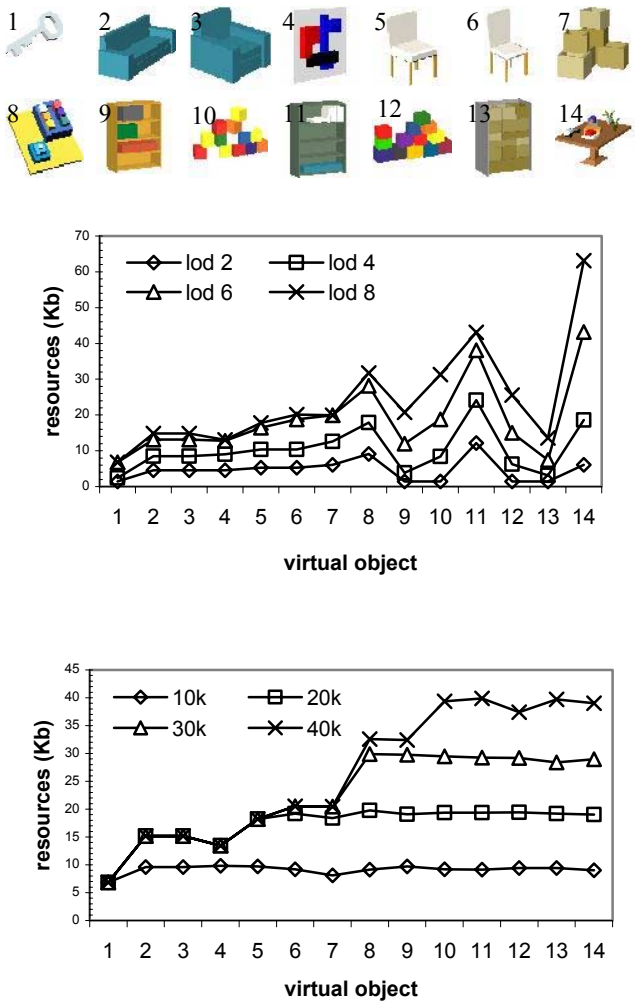

Figure 8. Network traffic for different documents: (a) Virtual objects used in the experiment. (b) Using rule-based mapping; (c) Using constraint-based mapping.

Each time a packet arrives, we sample the round-trip time (*SampleRTT*) and compute the average round-trip time (*AverageRTT*) using a running average:

$$AverageRTT = \alpha \times AverageRTT + (1 - \alpha) \times SampleRTT$$

$$UpdateRate = \frac{1}{AverageRTT}$$

The server will only transmit information to the client at a maximum rate of *UpdateRate*. Since some operations may be considered transient (i.e., whose effects are not final and are overridden by a subsequent operation), those arriving faster than *UpdateRate* can be discarded in order to reduce the effects of high latency. In the case of optimal simplification described in Section 4.3 above, this adaptation is performed on the mobile client instead of the server, when retrieving scene updates. This avoids the transmission of unnecessary updates over the wireless link. Figure 10 graphs the latency of a sample virtual world navigation in a mobile device,

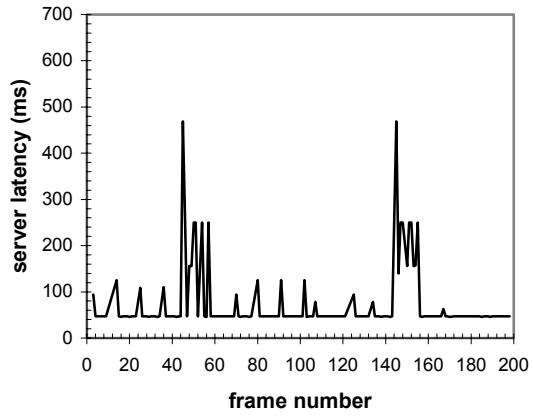

Figure 9: Server latency for optimal simplification of the virtual world in Figure 5 (total size ~ 8 MB). The simplified frames are shown on a mobile PDA device as the user navigates the virtual world.

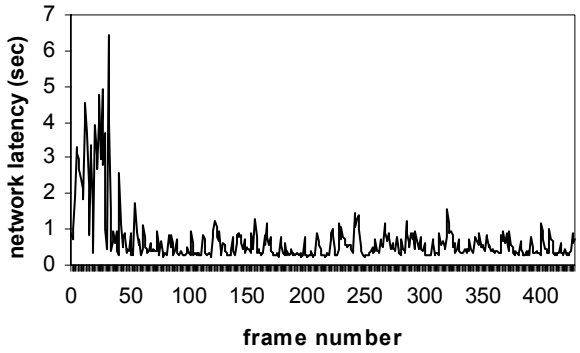

Figure 10. Network latency on a mobile PDA device connected via an IEEE 802.11b wireless local area network, navigating the virtual world in Figure 5 (*UpdateRate* is adapted using $\alpha = 0.9$).

using adaptation with $\alpha = 0.9$. Note that after a few initial iterations, latency is reduced to an average of 518 milliseconds.

6. CONCLUSIONS

We have developed a framework that manages heterogeneity in collaborative systems using customizable graph mappings. Graph mappings are obtained through a combination of topological mappings, used to simplify the shared data-structure, property mappings and user-defined policies. The key function of the framework is providing consistent application states to the different participants, under different aspects of heterogeneity: disparate device capabilities, user's interest and application domain differences. We showed that in addition to providing consistency, our framework can be used to guarantee resource bounds, such as limited bandwidth, memory usage and computing power, which are critical in mobile devices. Another important issue with collaborative systems is latency, which usually represents a tradeoff with the optimal management of system resources. We introduced two techniques to reduce the latency: (1) spatial coherence can be exploited to reduce the computation time of optimal mappings,

which in turn reduces the overhead latency imposed by our framework, and (2) bounding the update rate of transmission using a running average of the delays reduces the network delay, which is particularly noticeable in wireless networks.

7. ACKNOWLEDGMENTS

The research is supported by NSF Contract No. ANI-01-23910, US Army CECOM Contract No. DAAB07-02-C-P301, and by the Rutgers Center for Advanced Information Processing (CAIP) and its corporate affiliates.

8. REFERENCES

[1] Birman, K. P. *Building Secure and Reliable Network Applications*. Manning Publ. Co., 1996.

[2] Bright, M. W., Hurson, A. R., and Pakzad, S. Automated resolution of semantic heterogeneity in multidatabases. *ACM Trans. on Database Systems* 19, 2 (June 1994), 212-253.

[3] Britton, K. H., Case, R., Citron, A., Floyd, R., Li, Y., Seekamp, C., Topol, B., and Tracey, K. Transcoding: Extending e-business to new environments. *IBM Systems Journal* 40, 1 (2001), 153-178.

[4] Ceri, S., and Widom, J. Managing semantic heterogeneity with production rules and persistent queues. *Proc. Nineteenth Int'l Conf. Very Large Data Bases* (Dublin Ireland, August 1993), 108-119.

[5] Cho, G., and Shaw, D. X. A depth-first dynamic programming algorithm for the tree knapsack problem. *INFORMS J. Computing* 9, 4 (1997), 431-438.

[6] Connolly, T., Begg, C., and Strachan, A. *Database Systems: A Practical Approach to Design, Implementation, and Management*. Addison-Wesley Publ. Co., Wokingham, England, 1996.

[7] Correa, C., Marsic, I., and Sun, X. Semantic consistency optimization in heterogeneous virtual environments. Rutgers University, CAIP Center, Technical Report CAIP-TR-267, September 2002. Online at: http://www.caip.rutgers.edu/disciple/

[8] Fox, A., Gribble, S. D., Chawathe, Y., and Brewer, E. A. Adapting to network and client variation using active proxies: Lessons and perspectives. *IEEE Personal Communications (Special Issue on Adapting to Network and Client Variability)* 5, 4 (August 1998), 10-19.

[9] Funkhouser, T. A., and Sequin, C. H. Adaptive display algorithm for interactive frame rates during visualization of complex virtual environments. *Computer Graphics (Proc. SIGGRAPH '93)* 27 (1993), 247-254.

[10] Johnson, D. S., and Niemi, K. A. On knapsacks, partitions, and a new dynamic programming technique for trees. *Mathematics of Operations Research* 8 (1983), 1-14.

[11] Knister, M.J., and Prakash, A. DistEdit: A distributed toolkit for supporting multiple group editors. *Proc. ACM CSCW'90 Conf. Computer Supported Cooperative Work* (Los Angeles, CA, October 1990), 343-355.

[12] Lum, W. Y., and Lau, F. C. M. On balancing between transcoding overhead and spatial consumption in content adaptation. *Proc. 8th Int'l Conf. Mobile Computing and Networking (MobiCom '02)* (Atlanta GA, September 2002), ACM Press, 239-250.

[13] Mani, I. *Automatic Summarization*. John Benjamins Publ. Co., Amsterdam/Philadelphia, 2001.

[14] Marcu, D. *The Theory and Practice of Discourse Parsing and Summarization*. The MIT Press, Cambridge, MA, 2000.

[15] Marsic, I., Sun, X., Correa, C., and Liu, T. Maintaining state consistency across heterogeneous collaborative applications. Rutgers University, CAIP Center, Technical Report CAIP-TR-264, March 2002. Online at: http://www.caip.rutgers.edu/disciple

[16] Mascolo, C., Capra, L., Zachariadis, S., and Emmerich, W. XMIDDLE: A data-sharing middleware for mobile computing. *Wireless Personal Communications: An International Journal* 21, 1 (Apr. 2002), 77-103.

[17] Mason, A. E. W., and Blake, E. H. A graphical representation of the state spaces of hierarchical level-of-detail scene descriptions. *IEEE Transactions on Visualization and Computer Graphics* 7, 1 (January-March 2001), 70-75.

[18] Moving Picture Experts Group web page. Online at: http://www.mpeg.org

[19] Sciore, E., Siegel, M., and Rosenthal, A., Using semantic values to facilitate interoperability among heterogeneous information systems. *ACM Transactions on Database Systems* 19, 2 (June 1994), 254-290.

[20] Sun, C., and Chen, D. Consistency maintenance in real-time collaborative graphics editing systems. *ACM Transactions on Computer-Human Interaction* 9, 1 (March 2002), 1-41.

[21] SyncML Initiative Ltd., SyncML—Mobile Data Synchronization Protocol. Online at: http://www.syncml.org