Web Services Management Network: An Overlay Network for Federated Service Management

Vijay Machiraju, Akhil Sahai, Aad van Moorsel
Internet Systems and Storage Laboratory
HP Laboratories Palo Alto
HPL-2002-234
August 21st , 2002*

E-mail: {vijaym, asahai, aad}@hpl.hp.com

management; service management, SLA, web services, web service management network

We introduce the architecture, object model, components, and protocols of a management overlay for federated service management, called Web Services Management Network (WSMN). WSMN targets management of web services that interact across administrative domains, and therefore typically involves multiple stakeholders (examples are business-to-business, service provider interconnections, help desks). The architecture is based on (implicit) SLAs to formalize relations across domains. It relies on a network of communicating service intermediaries, each such intermediary being a proxy positioned between the service and the outside world. WSMN also exchanges control information to agree on what to monitor, where to monitor, and whom to provide visibility.

* Internal Accession Date Only

Approved for External Publication
© Copyright Hewlett-Packard Company 2002

Web Services Management Network: An Overlay Network for Federated Service Management

Vijay Machiraju, Akhil Sahai, Aad van Moorsel

Hewlett-Packard Laboratories
1501 Page Mill Road, Palo Alto, CA 94034
{vijaym, asahai, aad}@hpl.hp.com

Abstract

We introduce the architecture, object model, components, and protocols of a management overlay for federated service management, called Web Services Management Network (WSMN). WSMN targets management of web services that interact across administrative domains, and therefore typically involves multiple stakeholders (examples are business-to-business, service provider interconnections, help desks). The architecture is based on (implicit) SLAs to formalize relations across domains. It relies on a network of communicating service intermediaries, each such intermediary being a proxy positioned between the service and the outside world. WSMN also exchanges control information to agree on what to monitor, where to monitor, and whom to provide visibility.

Keywords: management, service management, SLA, web services, web service management network

1 Introduction

By packaging software applications as 'services' that are accessible over the Internet or intranet, enterprises achieve new and better means to utilize their own and each other's applications. Services[1] can be accessed through manual user activities (e.g., browser-based IT help desk), and increasingly through other services. In the latter case, conglomerations of interacting services emerge, which access one another through more and more automated means. Examples can be found in business-to-business computing (web services, supply-chain processes, payment services), service providers (utility or grid computing) or in enterprise applications (payroll applications, remote IT services).

In the emerging world of Internet services, operational management becomes exceedingly important and challenging (and thus interesting). Services often directly impact the business process execution, and mishaps may directly be reflected in the bottom line of a business. This puts a premium on fault and performance management capabilities for services. Moreover, the services paradigm increases the complexity of run-time operations. Services communicate across monitoring domains [1], include different business partners, and are likely to rely on third parties to complete a service offering. As a consequence, service management has to deal with multi-party interactions, has to collect a large amount of data and synthesize it to understand the health of relationships between partners, and must resolve the limited end-to-end visibility and control one has over each other's services.

[1] We use the terms 'service' and 'web service' interchangeably, but prefer the term 'service,' since it stresses that we discuss the management of applications exposed as *services*, instead of the fact that we assume these services to communicate through *web services* technology (SOAP, XML, WSDL).

From the above we conclude that traditional application management must evolve into (or be complemented by) ‘true’ service management, and ultimately into service ‘relationship’ management. First, we must manage a service ‘as a whole,’ that is, as it is provided to a partner. Contrast this with application management, for which it is necessary to understand how the service is internally implemented through a set of objects and what exceptions each object generates. Instead, ‘true’ service management manages the interactions a service has with other services or consumers, for which we need visibility at the service interface between an application and its users. Secondly, we must manage relations, not only through local management, but also through sharing data between partners as needed, and, more importantly, through exchanging signaling information about monitoring, service levels and control actions. In other words, we need to be concerned with federated management [2].

In this paper, therefore, we propose Web Service Management Network (WSMN), a management architecture for federated service management. Since we believe it is safe to assume Internet services will be implemented using web service technology (SOAP, XML, WSDL), WSMN is based on such technology. The critical concept in WSDN is that of SLAs. If SLAs are explicitly defined we adapt them to make them manageable, and if no SLAs are actually agreed upon between services, WSMN manages towards SLAs imposed specifically for management purposes (‘implicit SLAs’). The SLA concept allows us to frame and solve many problems rather elegantly and effectively, as we discuss further in Section 2.1, and illustrate using a WSMN prototype implementation in Section 4. WSMN, then, is a network of cooperating intermediaries, each such intermediary implemented as a proxy sitting between a service and the outside world, and a set of protocols to manage service relationships expressed through SLAs.

One can regard WSMN as a logical signaling network for management purposes, a concept well known from telecom (SS7 [3]). In fact, throughout the various layers of the Internet stack, overlays are being proposed to establish quality guarantees that the Internet itself cannot create. Examples exist for instance for Internet telephony (SIP [4]) and streaming media content delivery [5]. Also of interest are the various emerging solutions to provide properties and features such as security, transactionality and change management to C2B and B2B web services. Examples are Flamenco Networks [6], Kenamea [7] and Talking Blocks [8]. All these companies use networks of collaborating intermediaries, often including a third-party play (repositories as well as services). None of these solutions addresses service management, as we do in this paper. Recently, Gartner surveyed and put in perspective these architectures, which they term web service networks [9]. Our term ‘WSMN’ is therefore extra appropriate, since our approach uses a web services network architecture for purposes of service management.

We believe that future web services management technology and standards must be based on WSMN as the architectural underpinning. The primary objective of this paper, therefore, is to introduce and explain WSMN and argue its value as core architecture for service and service relationship management. To this end, we introduce the main concepts behind WSMN in Section 2 (SLAs and protocols, respectively) and discuss the details of the intermediaries in Section 3. Section 4 then demonstrates multi-party SLA management using a WSMN prototype implementation.

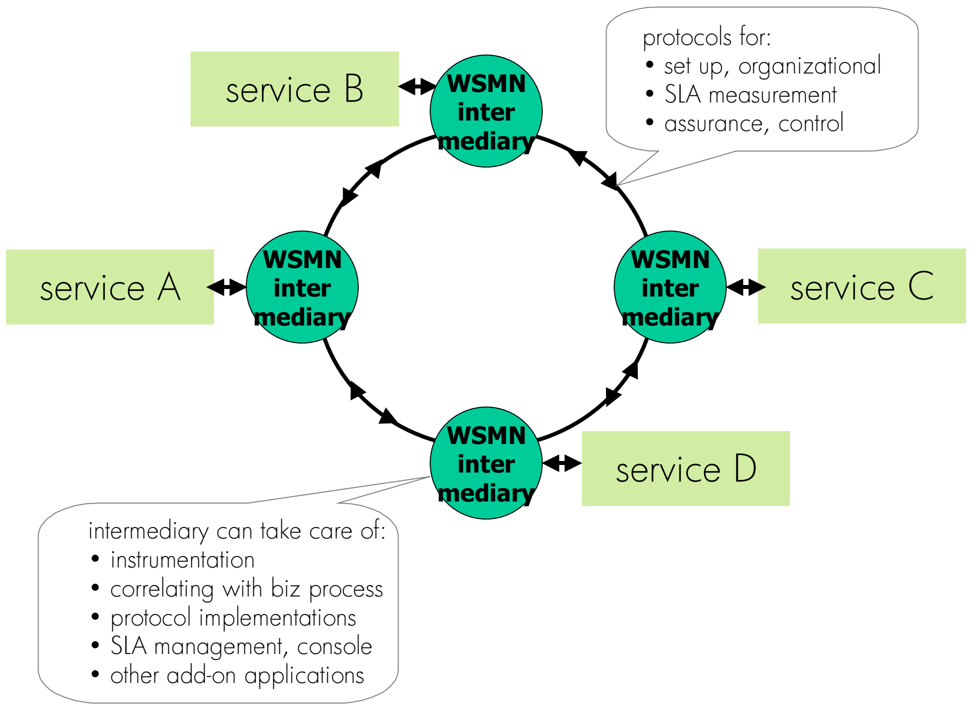


Fig. 1 Web Services Management Network

2 WSMN Design Choices

In one sentence, Web Services Management Network is a logical overlay network for SLA management between services, constituted of communicating intermediaries. Fig. 1 illustrates WSMN, each intermediary being a proxy between a service (providing the interface to one or more applications) and the outside world. The most critical and interesting aspects of our design are (1) the choice to base all management on SLAs, and (2) the protocols for intermediaries to collaborate. We will argue in detail why we made these two design decisions, and discuss their consequences for the components in the intermediaries. We will not further elaborate why service management logic is placed in intermediaries, since we think that is an obvious enough choice—the reader can find a discussion in [9] and [10]. We note that we assume that services interact using web services technology, that is, XML, SOAP and WSDL [11].

2.1 SLAs as a Management Tool

SLAs are, of course, well-established in management [12], and one can argue that especially in service relationship management, SLAs will be of increasing and singular importance [10] [13]. However, our primary incentive to use SLAs is different: it is not that the existence of SLAs introduces a management problem we must deal with, but that voluntary introduction of SLAs can be a tool to manage service relationships. Hence, if SLAs are not sufficiently detailed, or are not explicitly agreed upon between services, we introduce SLAs for the purpose of management. We call these SLAs 'implicit SLAs.'

A service level agreement defines a basic abstraction through which partners can understand each other’s capabilities, negotiate on service parameters, and manage to agreed levels. SLAs are a clean way to separate management concerns between the partners. In a situation where multiple partners interact with each other to accomplish a goal/task, an SLA is defined between every pair of interacting partners with well-defined expectations. This allows management of the overall task to be broken up up-front into smaller problems of managing each interaction. An alternate architecture is one that requires centralized intelligence, which makes sure that the overall task completes. However, such an architecture requires management data to be massively shared between all partners and is not scalable to a large number of partners. Moreover, in situations where each of the managed services runs within its own management domain, a centralized architecture will not be feasible.

Many problems in service relationship management can be solved through SLA management. For example, the problem of discovering and selecting right partners translates to the problem of querying for or negotiating an SLA. Similarly, the problem of managing a relationship translates to the problem of monitoring and assuring an SLA. Finally, solutions for rating partners and optimizing partnerships can be built by optimizing SLAs based on the cost of meeting them and penalties of not meeting them. Due to this close association, we use the words SLA management and relationship management interchangeably in the rest of this paper.

In WSMN, once an intermediary learns about an SLA (either input through a console or as the result of a semi-automated SLA negotiation protocol), it determines what elements to monitor, at what intermediary to monitor them, and how often to set alarms (see also Section 4). To establish this automated SLA management, the SLA must be specified unambiguously. In Appendix B, we represent the SLA specification language we developed for that purpose. This specification language relies on a managed object model for web services presented in Appendix A.

2.2 Protocols

The intermediaries interact amongst each other through a set of management protocols. These protocols range from basic ones that involve establishment and sustenance of the network to higher-level protocols that implement higher-level functionalities (partner selection, SLA assurance). We distinguish three classes of protocols: life-cycle protocols, measurement protocols and assurance protocols.

Life-Cycle Protocols. The life-cycle protocols are basic protocols that deal with initiation and sustenance of the WSMN. In our implementation, the initiation protocols are commenced as soon as two web services start communicating with each other. It is assumed that if the web service supports WSMN, the intermediaries are reachable at the address obtained by conjugating ”/WSMN” at the end of the service URL. The intermediary that detects a web service communication initiates an establishment protocol that acquaints the intermediaries of each other’s capabilities. After the establishment phase, the synchronization protocol is executed. A reasonably accurate clock synchronization protocol is needed because out of sync clocks could lead to erroneous results. Once a WSMN is established, unless it is torn down through an explicit teardown protocol, the intermediaries exchange keep-alive messages as part of the keep-alive protocol. Through all these protocols, the WSMN manages the various phases of its life cycle.

Measurement Protocols. We feel it is important to argue why service management requires measurement protocols between intermediaries, because it is not necessarily obvious such protocols are required. The key question is, if party *A* guarantees an SLA to party *B*, why is it not sufficient to monitor the compliance to this SLA at the intermediary of *A*?

There exist scenarios in which not all data required for managing SLAs can be measured locally. For example, consider a web service *A* that promises certain goods to be delivered to a customer *B* within five days. However, the web service uses a third service *C* for shipping the goods. As a result, *A* will not know when the goods were in fact delivered to *B*. The only way that *A* can measure its SLA is by consultation with *B* or *C*. We will see this example arise in the prototype discussion in Section 4. There, *A* monitors order arrival, *B* monitors delivery, and they exchange the results using the measurement exchange protocol. Another common example involves, say, a payment service provider *P* promising a certain perceived (end-user) transaction throughput to its customers. However, there are several service providers in the flow of execution between a customer and *P* – *P*'s data center *Q*, *Q*'s Internet service provider *R*, *R*'s carrier service provider *S*, and customer's Internet service provider *T* – all of which have an influence on the perceived throughput.

A common theme in all these examples is that the SLA dependency relations between web services do not exactly match the execution dependencies (in the first example above, *A* has an SLA dependency with *B,* but the execution dependencies also include *C*). An SLA can exist between two partners even though there are other players that influence the outcome of the SLA. Hence, SLA monitoring between web services requires an infrastructure where limited management information can be exchanged between partners in a trusted and secure manner. For this purpose, we developed a measurement exchange protocol, the details of which are described in Section 3.2.

Assurance Protocols. Assurance protocols are higher-level protocols executing more complex interactions, related to run-time optimization and control of SLAs. SLA negotiation requires negotiation protocols to be executed between the intermediaries. SLA assurance can be done by dynamically changing partners with different levels of service as suppliers. In another example, we developed a trouble-ticket exchange protocol, which forwards warnings between intermediaries as soon as SLAs are not met or are in danger of not being met. These trouble tickets are created, their status checked and are closed once the problem is satisfactorily fixed. Assurance protocols may very well depend on additional services, possibly offered through third parties—e.g., a UDDI repository helps discovering web services, third-party negotiation services help setting SLAs, rating services (which maintain records on web service performance) help identifying the right partners, etc. This paper does not focus on such assurance protocols.

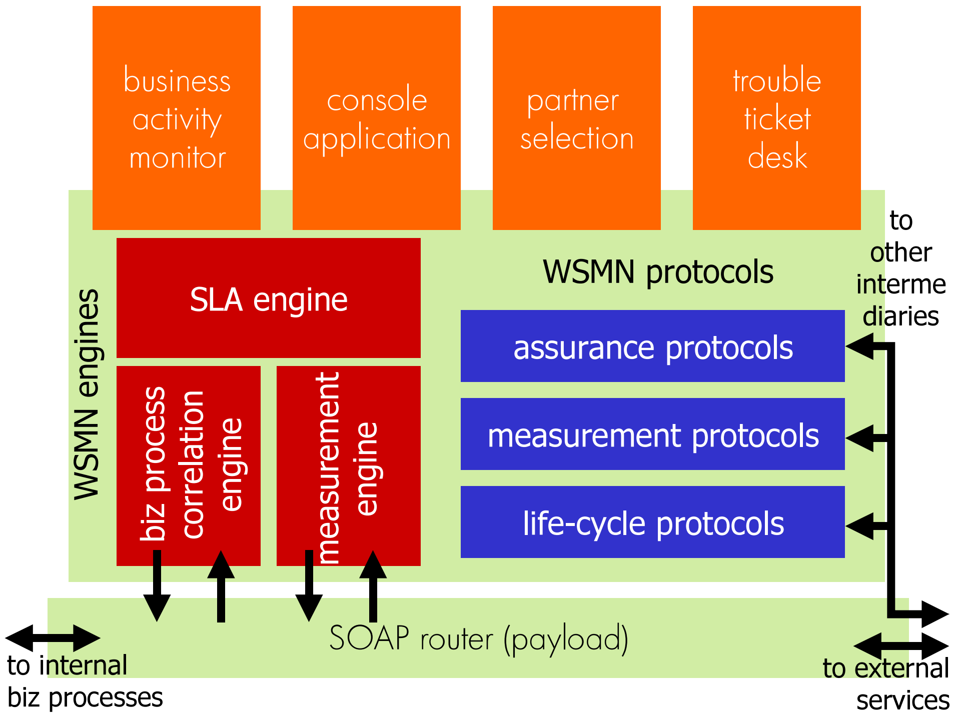


Fig. 2 Components of WSMN intermediary

3 Monitoring at the Intermediaries

Fig. 2 illustrates the various components of the intermediary. The intermediary is embedded in the SOAP router, and has three main groups of components: (1) WSMN engines for measurement and SLA management, (2) WSMN protocol implementations, and (3) applications exploiting the engines and protocols. We discuss the first two items in detail, in particular with respect to instrumentation issues, and refer to Section 2 as well as the example in Section 4 for various applications that utilize WSMN.

3.1 Monitoring Engines

SOAP routers receive the messages from SOAP clients and submit them to the receivers (the *payload* layer in Fig. 2). SOAP routers are the obvious candidate to support the intermediaries, since they already act as proxy for the web service interactions between a collection of services and the outside world. The WSMN intermediaries add management capabilities to SOAP routers, by capturing SOAP messages, potentially modifying the SOAP headers and extracting information from those messages. This can be done without any modifications to existing applications, and without re-compilation of the existing SOAP toolkit installation.

Fig. 2 demonstrates two measurement engines that intercept the SOAP messages and manipulate them for measurement purposes: *measurement engine* and *business process correlation engine*. The measurement engine deals with measuring the interactions with the outside world, while the business process correlation engine deals with measuring the interactions with a process engine that maintains the state of conversations with partners (as specified through standards such as

RosettaNet or ebXML). The arrows between the *SOAP Router* and the *WSMN engines* signify that the engines intercept the SOAP message and detour the control flow.

Both measurement engines utilize a message tracking protocol, which allows one to correlate delays and other information over all segments a transaction traverses [14] [15]. A notion of global flow is introduced by this protocol. Messages in the same global flow use a GUID as identifier either as defined in some protocols (RosettaNet) or injected in SOAP headers by the measurement engine. The measurement engine checks every time it catches a message whether a GUID is present, and, if no GUID exists, it inserts a GUID into the SOAP header of the message. All SOAP routers propagate the GUID in their communications, and, consequently, all intermediaries are able to figure out which SOAP message is sent in the context of which previous messages. The details of the message tracking can be found in [14] and extensions to deal with the process engine are explained in [15].

In addition to the message data, one may very well be interested in gathering other information about the business, and correlate the activities with external message exchanges. To provide for that, one needs to add an application to the intermediary, such as the *business activity monitor* in Fig. 2. For example, HP Process Manager logs execution data into a raw file, which can then be uploaded into database tables by the dedicated application. Alternatively, the add-on application uses the Java API provided by HP Process Manager. In combination with the GUID-based message tracking this provides a rich business activity monitoring solution [16].

3.2 Measurement Exchange Protocol

All WSMN protocols use web services transport, that is, the WSMN messages pass through the SOAP router just like messages with execution payload. However, as the arrows in Fig. 2 indicate, WSMN messages do not ‘merge’ with the payload (as with interception methods), but are treated independently.[2] We assume that the necessary life-cycle protocols have executed as described in Section 2.2., to establish a well-functioning WSMN. As we argued in Section 2.2, it may then be required to combine the measurements of various intermediaries to determine if an SLA has been met. Therefore, we developed the measurement exchange protocol.

The measurement exchange protocol has been designed with the following objectives in mind: (a) minimize the amount of data that is transmitted between the two intermediaries, and (b) transfer the data in time for the evaluation of SLA to take place when triggered. Based on these two goals, WSMN intermediaries must agree on (a) what measurements need to be transferred and at what level of aggregation, and (b) how frequently must they be transferred. This is determined by the SLA specification. The details of the SLA specification are given in Appendix B, but the important attributes are *evalFunc*, *evalWhen* and *measuredAt*. *measuredAt* specifies which service (and thus which intermediary) collects the data, and *evalWhen* specifies at what moments in time to collect the data. The attribute *evalFunc* allows us to be smart in aggregation of the measurements, using typical sampling functions such as *count (t)*, *totaled*, *averaged*, *movingAvg(lastN)*, *minN*, *maxN*, *threshold* (see [17] for possible strategies in data aggregation). In the case when the sampling function cannot be determined from the *evalFunc*, we transfer all the measurements from one side to the other.

[2] Note that the three protocol types (*life-cycle, measurement* and *assurance*) in Fig. 2 are just a classification and do not form a stack in the sense of the ISO reference model. All WSMN protocols execute independently from each other, but have in common that they communicate using SOAP (that is, taking the stack perspective, all WSMN protocols sit one layer above SOAP).

The resulting measurement exchange protocol makes sure that there is agreement on the level of aggregation and the frequency of transferring data. This results in five different types of messages, which together form the protocol. We explain the primitives in terms of a scenario in which a ‘provider’ obtains data from a ‘customer’ (see Section 4). The primitives are:

- *Init*: sent by the customer to the provider for clauses whose measurement data need to be exchanged. The *init* message carries possible choices of sampling function, interval, duration and reporting interval details that the consumer supports.
- *Request*: The provider decides the exact measurement specification (sampling function, sampling parameters and reporting parameters) that it chooses and specifies it in its request message.
- *Agreement*: The customer sends this message if it agrees to the request
- *Start*: message from provider to commence the reporting
- *Report*: actual measurement report messages
- *Close*: message to terminate the reporting

4 Application of WSMN

To demonstrate and test WSMN, we built a prototype WSMN intermediary, and created a test environment based on a business-to-business scenario. In this scenario, a PC vendor wants to manage SLAs with its customers and suppliers. Fig. 3 shows the various players in this scenario: the vendor *PCMaker*, its supplier web services *ChipSupply*, *Assembly*, *Payment*, *Delivery* and a customer service, namely *PCBuyer1*, thus resulting in a WSMN with six intermediaries. We have no illusions that this scenario is particularly close to reality, but it serves our purposes of demonstration and testing. A typical message exchange sequence between the various players is shown in Fig. 4. Effectively, the (potential) buyer logs in with the vendor and asks for a quote from the PC vendor. *PCMaker* first checks with one of its suppliers and then returns a quote to the buyer. In this case, the buyer decides to order, and the PC vendor executes the order through its providers. Note that no doubt a lot of manual work is involved at various stages, but that the interactions in Fig. 3 and Fig. 4 only refer to electronic messages. *PCMaker* agrees on an SLA with the buyer, stipulating that delivery will not take more than some number of days—the two parties agree in their SLA that this corresponds to the time from the moment that *PCMaker* receives order message 7 until *PCBuyer1* acknowledges delivery through message 16.

Executing life-cycle protocols to set up WSMN. The WSMN is created as soon as the service to service communication is detected by the intermediaries. For that, we implemented the life-cycle protocols mentioned in Section 2.2. We also implemented a third-party synchronizer service. This service is used by all the intermediaries to synchronize their clocks.

We added a console as an add-on application to the intermediaries (as denoted in the second application box in Fig. 2). Later we see how we use that for visualizing various aspects of SLA management, but we also use it to show what services are running. In Fig. 5 we see the console for *PCMaker*. It depicts the services that are known to *PCMaker*, resulting in an up-to-date ‘run-time version’ of Fig. 3. The latest interaction that the measurement engine intercepted is colored blue—as one can see, the latest communication at the moment of this snapshot was with the synchronizer service.

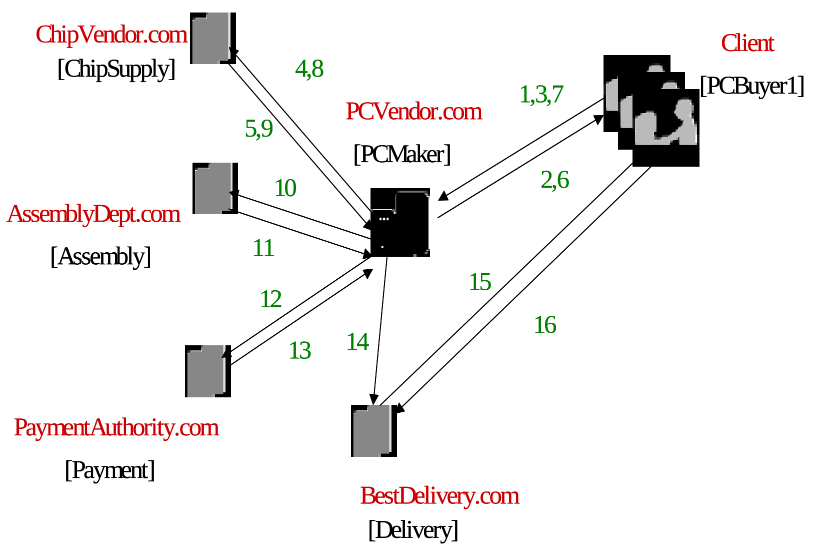


Fig. 3 Example interactions

#	MSG_TYPE	SENDER	RECEIVER
1	SubmitLoginmsg	PCBuyer1	PCMaker
2	ConfirmLoginmsg	PCMaker	PCBuyer1
3	SubmitQuoteRequestmsg	PCBuyer1	PCMaker
4	RequestChipQuotemsg	PCMaker	ChipSupply
5	SendChipQuotemsg	ChipSupply	PCMaker
6	SendQuotemsg	PCMaker	PCBuyer1
7	SubmitPORequestmsg	PCBuyer1	PCMaker
8	SendChipPOmsg	PCMaker	ChipSupply
9	RespondChipPOmsg	ChipSupply	PCMaker
10	SendAssemblyPOmsg	PCMaker	Assembly
11	RespondAssemblyPOmsg	Assembly	PCMaker
12	SendPaymentPOmsg	PCMaker	Payment
13	RespondPaymentPOmsg	Payment	PCMaker
14	SendDeliveryPOmsg	PCMaker	Delivery
15	SendDeliveryNotificationmsg	Delivery	PCBuyer1
16	SendReceiptNotificationmsg	PCBuyer1	Delivery

Fig. 4 Interactions for the example

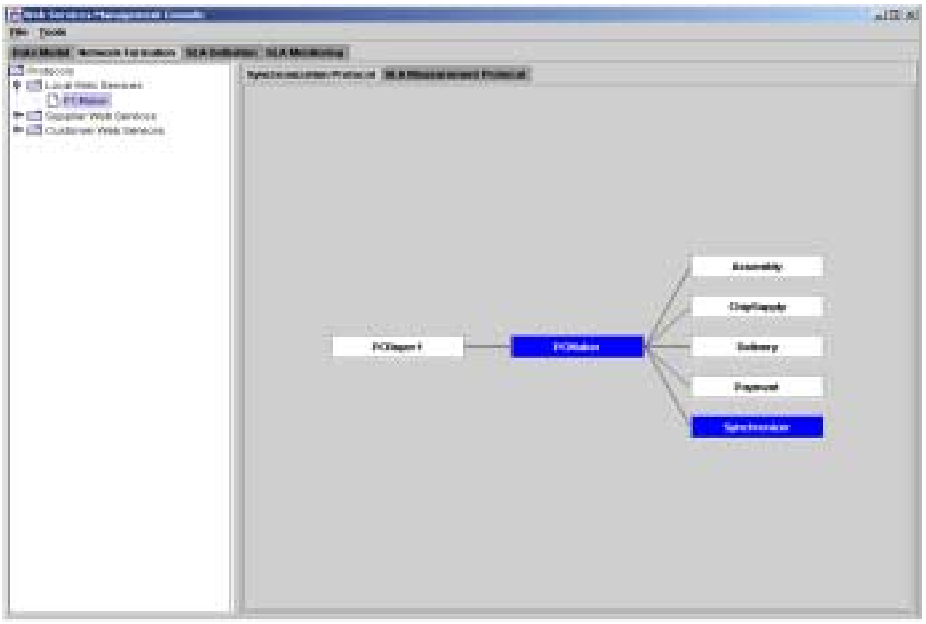


Fig. 5 Visualization of clock synchronization protocol execution

```
<sla>
  <slaId>3</slaId>
  <startDate>Fri Feb 15 00:00:00 PST 2002</startDate>
  <endDate>Mon Jul 15 00:00:00 PDT 2002</endDate>
  <slo>
    <sloId>1</sloId>
    <dayTimeConstraint>Mon-Fri: 9-17</dayTimeConstraint>
    <measuredItem>
      <item>
        <constructType>message</constructType>
        <constructRef>PCMaker.com/SubmitPORequestmsg</constructRef>
        <measuredAt>PCMaker.com</measuredAt>
      </item>
      <item>
        <constructType>message</constructType>
        <constructRef>PCBuyer1.com/SendReceiptNotificationmsg</constructRef>
        <measuredAt>PCBuyer1.com</measuredAt>
      </item>
    </measuredItem>
    <evalWhen>6PM</evalWhen>
    <evalOn>all</evalOn>
    <evalFunc name =”averageResponseTime”  operator =”LT” Threshold =”5”
                unit =”days”></evalFunc>
  </slo>
</sla>
```

Fig. 6 XML specification of an SLA

Setting up an SLA. In order to demonstrate SLA management in the overlay network we defined an SLA between *PCMaker* and *PCBuyer1*. Fig. 6 provides the details of one agreed upon SLA, in XML format: over the specified period, the average time from when *PCMaker* receives the order until *PCBuyer1* acknowledges its delivery, must be less than five days (this is the time between message 7 and 16 in Fig. 4). The SLO in the agreement requires measurement at each end-point. The WSMN intermediaries at *PCBuyer1* and *PCMaker* utilize the measurement exchange protocol to agree on sending measurements for *SendReceiptNotificationmsg* everyday just before 6 PM and keep sending the reports from *startDate* to *endDate*. In our prototype, the console allows one to load the SLA in the intermediary at both at *PCBuyer1* and *PCMaker*—once loaded, the intermediary immediately starts the processes necessary for SLA management.

Implementation of the intermediary. All services in our prototype have been implemented using Apache SOAP toolkit, and we extended the SOAP toolkit to collect the message correlation and instrumentation data. We use WSFL [11] as flow language, use HP Process Manager (HPPM) for orchestration of conversations between services. Since HPPM provides a Java API to control process executions by other software components, the business process correlation engine can use this API to feed the GUID into HPPM process instances and retrieve it when necessary. The measured data is all stored and modeled in a mySql database for short-term storage and in an Oracle9i data warehouse for long term archiving. Fig. 7 illustrates these and other components; notice that Fig. 7 is a more detailed version of Fig. 2.

The *model generator* in the intermediary receives the WSDL/WSFL specifications and creates a model of the web service in the *model repository*. All the measurements collected from the service (e.g., ongoing conversations, performance measurements, etc) are attached to this model. The instrumentation in the web service is responsible for collecting these measurements and passing them on to the *measurement handler* to be stored in the model repository. If the measurements are collected on the client side (as determined by the *measuredAt* components of the items in SLA clauses), then the *communicator* is responsible for receiving the measurements and storing them into the repository. If it is required that management data is transferred between intermediaries, then the *management protocol handler* executes the measurement exchange protocol (see Section 3.2).

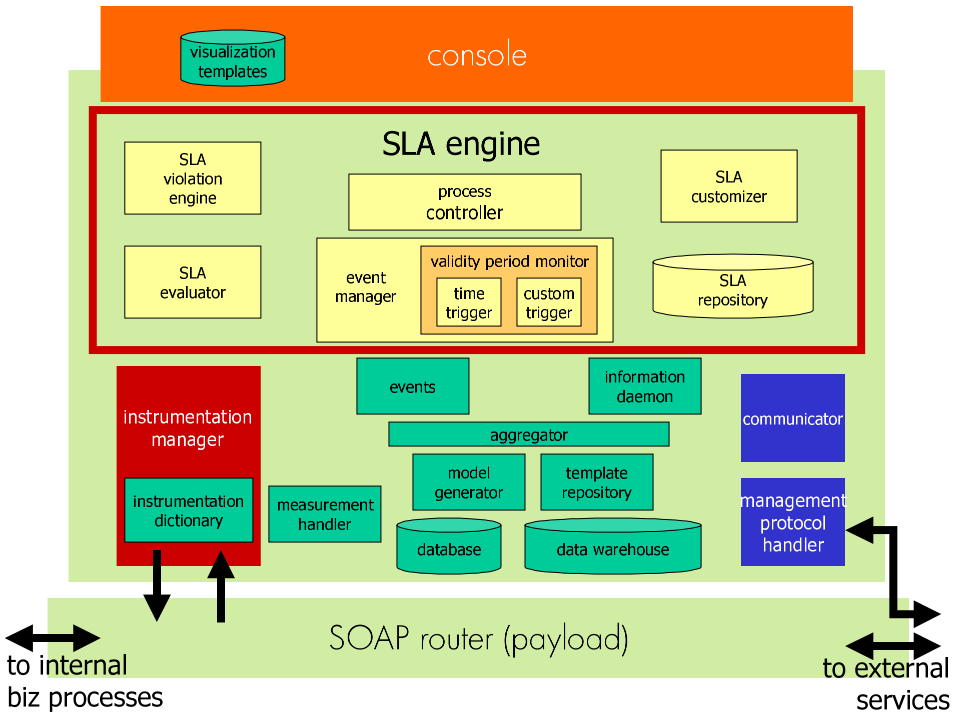


Fig. 7 Components of the WSMN intermediary (detailed version)

SLM engine. As soon as the SLM engine *management process controller* receives the SLA it executes a *monitoring process flow* and accordingly informs the *SLA customizer*, which in turn customizes the alarms at the *Event Manager* (depending on the *evalWhen* and *dateconstraint* components). The *Event Manager* comprises of the *SLO Validity Period Monitor* and triggers (time based and event based). The *SLA customizer* also creates an *SLO object* in the *SLA repository* and registers it as the call back handler of the events. The *SLO object* maintains the state of the SLO (*valid*, *active*, *invalid*). If a registered event for start-date of an SLO arrives the state of the SLO is changed from *init* to *valid*. The SLO is invalidated when the *end-date trigger* arrives. While the *evalWhen* events are triggered (because of a time or an event happening) the *SLO evaluator* evaluates the SLO. The *SLO evaluator* obtains the required management information (based on *evalOn*, *daytime constraint* and the *evalFunc* constituent of the specification) from the high performance database in memory. The *SLO evaluator* determines compliance/violations. The *SLA violation engine* maintains the record for violations, their timestamps, the levels of violation, and the clauses that are violated (both in memory and in log files). The violation records will also be used by the *SLA violation engine* for triggering actions specified by *evalAction* constituent of the SLO.

Management console. The management console is an add-on application, and can be used for a variety of purposes, such as inputting SLAs into the engine, visualizing the current SLAs/SLOs, their violation records and to analyze the log files. Two snapshots of the intermediary console for *PCMaker* are shown in Fig. 7 and Fig. 8. Fig. 7 shows a screen with SLA monitoring data, apparently for SLA number 2 with *PCBuyer1*. Fig. 8 shows some more advanced visualization of SLA violations of the same SLA, for *SLOid* 1 (the visualization technique is a pixel bar chart [18]).

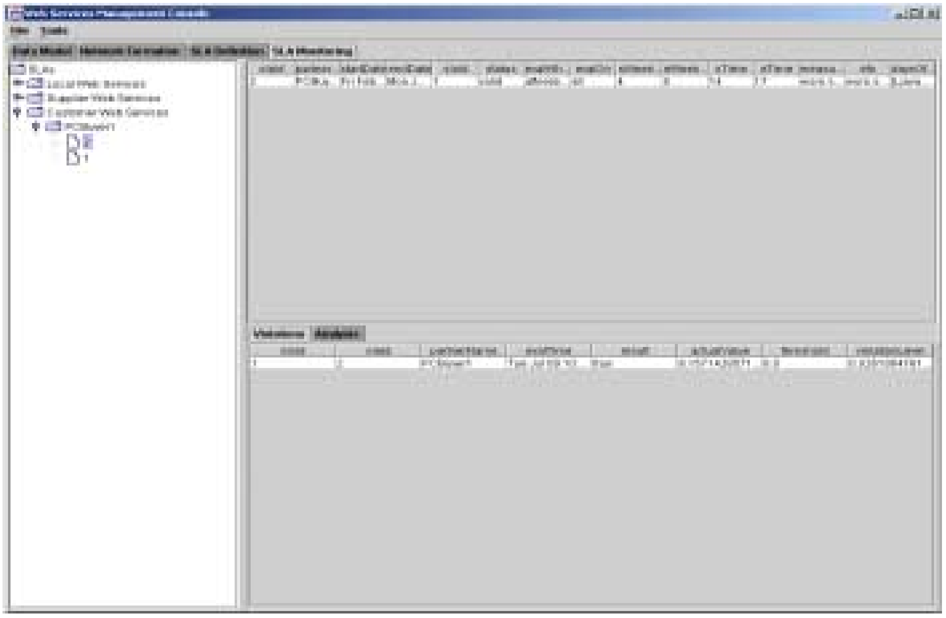
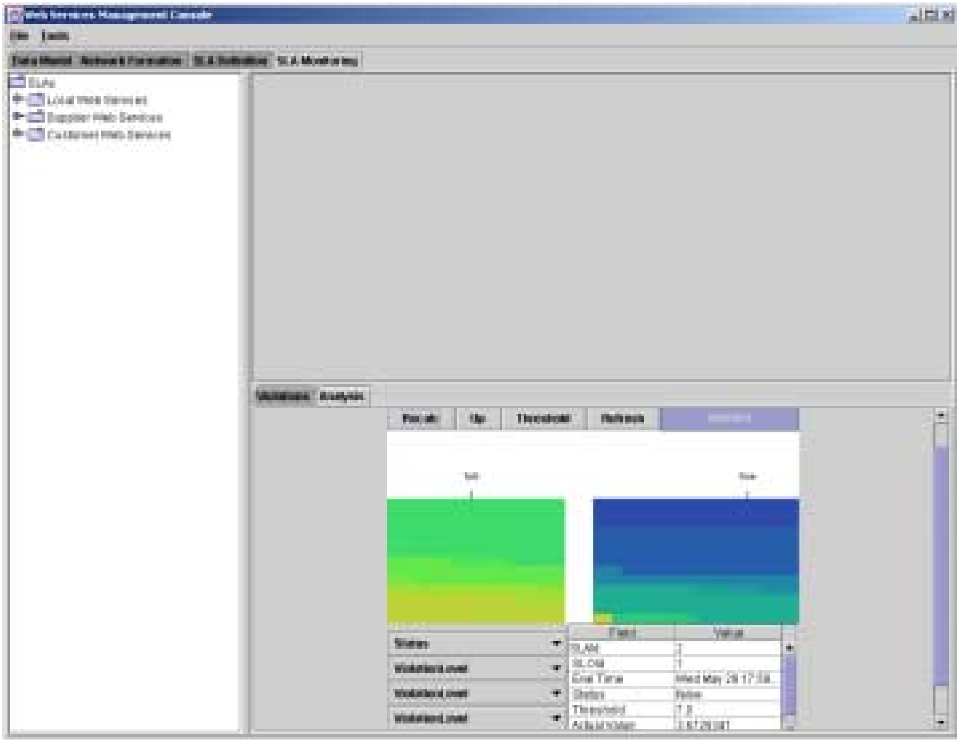


Fig. 8 Console for intermediary: monitoring SLAs



Field	Value
SLAId	1
SLOId	1
End Time	Wed May 28 17:58
Status	false
Threshold	7.0
Actual Value	3.6729 [illegible]

Fig. 9 Visualization of SLA violations

5 Conclusion

WSMN provides the appropriate architectural underpinnings for future development in web service management technology and standards. WSMN is a logical overlay network constituted of communicating intermediaries, each such intermediary implemented as a proxy sitting between a service and the outside world. It assumes a service-centric model for application usage, and focuses on managing the service offering (as opposed to the internals of applications). Moreover, it realizes that service management will be more and more about managing the overall quality of interactions (service relationship management), and uses the concept of SLAs to provide a flexible and scalable management solution for such management.

‘Implicit SLAs’ are introduced, to allow for management of SLAs that have not explicitly been agreed upon or of SLAs that do not specify enough monitoring details. WSMN also provides a set of protocols necessary to deal with cases in which multiple parties must share management data to evaluate if SLAs are successfully met. Future research must deal with operational aspects of WSMN, including issues of security and trust (along the lines of the work in [14] for message tracking). The existing prototype implementation of WSMN allowed us to demonstrate the workings of the overlay network, and will help to further test the appropriateness of our current and future design decisions.

Appendix A Web Service Management Object Model

Core to every management solution is the underlying object model necessary to identify what system aspects can or must be monitored. For web services management, other than OMI [19],no standard object models are available yet, but the models behind standards such as WSDL, WSFL and ebXML provide us with the core constructs necessary for a web services object model. In this paper, we do not provide details of the object model we used. However, since the object model is input to various pieces of the WSMN, we graphically display a part of the used object model in Fig. 11. Together with definitions in Fig. 10 of some of the terminology, we hope this is enough information to make this paper self-contained. The details of the model are published as a technical report in [20].

Business	An organization that executes business processes. The business marks the boundaries of an administrator’s *domain of responsibility* . A business can put out one or more service providers. A service provider controls its Business Process Flows.
ProcessFlow	A sequence of one or more workflow *activities* that achieve some intended purpose on behalf of the business.
Activity	Logical entities that form a workflow. Is realized by one or more applications and exposed as one or more operations
Application	Implements an activity.
Operation	Exposed part of the activities in a WSDL description
Message	An Operation is made up of one or more messages
User	A specific business, which invokes operations. A user could be a service provider too in a B2B scenario.
Service provider	A service provider provides services and Business Process Flows.

Fig. 10 Definition of terms

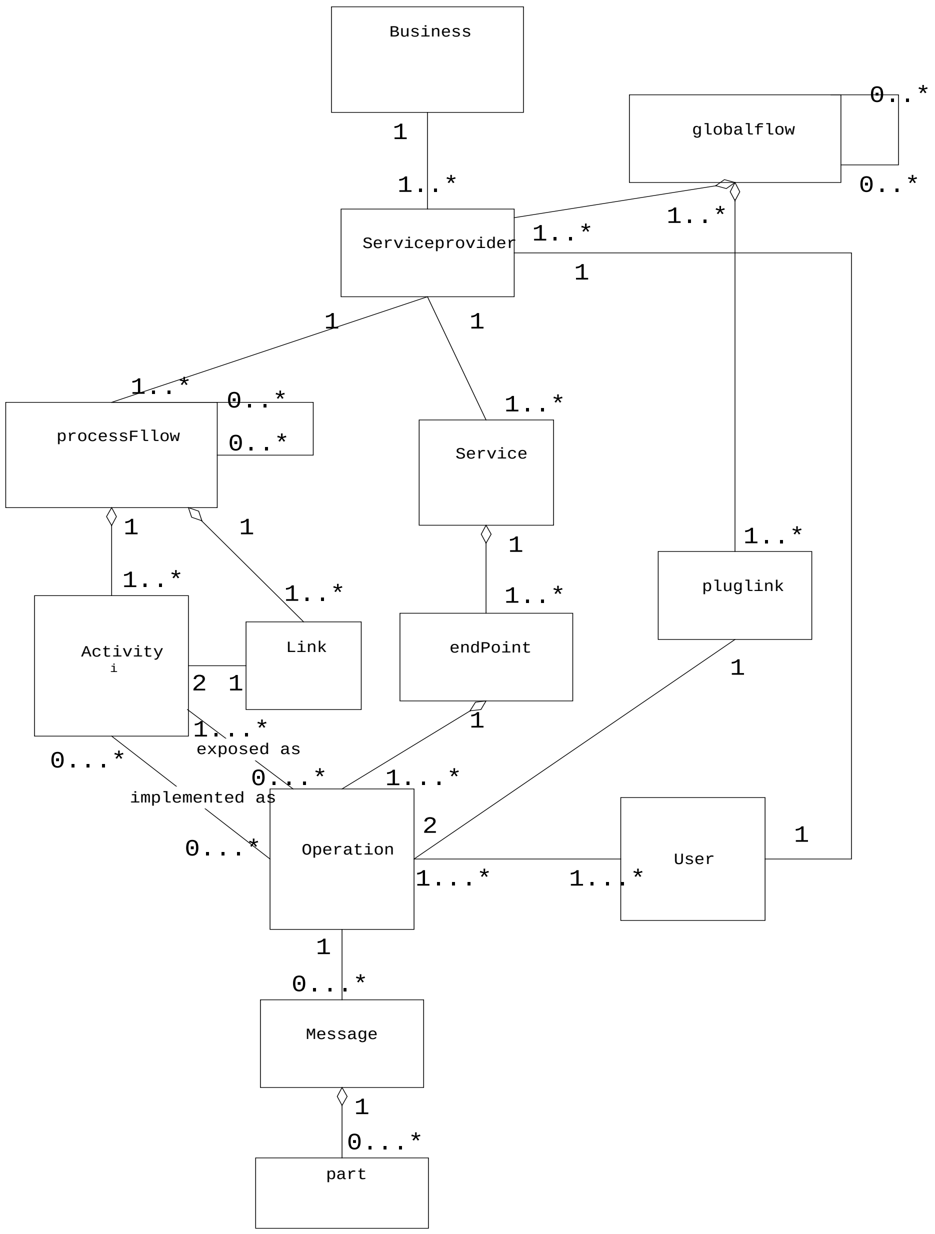


Fig. 11 Object model

Appendix B SLA Specification

SLAs are central to WSMNs, and we therefore spent considerable effort in defining an appropriate SLA definition. To allow semi-automatic SLA management, and multi-party SLA management, one must have unambiguously defined SLAs at the right level of granularity, agreed upon across parties. Since existing languages were not tailored to the way we use SLAs, we created our own language, which serves the purposes of our work [21]. Note that we built our SLA formalization upon the object model discussed in Appendix A (which in turn builds on WSDL, WSFL and ebXML notions).

In order to formalize the SLA specification, we looked at the existing SLA/Contracts [12] and envisaged a typical contract between a company manufacturing PCs (say *PCMaker*) and a company buying PCs (*PCBuyer*) for a period of 6 months. The kind of service level objectives (SLOs) we want to express are: "*PCMaker* shall deliver the ordered goods on an average within 10 days of the receipt of a purchase order," or "PCMaker shall invoice PCBuyer for any goods ordered *within10 days*."

An SLA is specified over a set of data that is measurable. An SLA has a date constraint (*start date*, *end date*, *nextevaldate*) and a set of SLOs. Each SLO also has a functional part (that refers to a *system*, *endpoint*, *process*, or a *set of processes*…) and a guarantee applied on the functional part. The guarantee is on a *system*, a particular instance of a construct (*process*/*operation*/*message*…) or on a *set of such constructs*. An SLO in turn has typically a *day–time* (Mon-Wed, 6:00PM-8:00 PM) constraint and a set of clauses that make up the SLO. A clause is based on measured data. This is referred to as a *measuredItem*. A *measuredItem* can contain one or more *items*. A *measuredAt* element determines where the measurements are taken (*provider, consume*). A clause evaluation is triggered either when an event happens, e.g. say a message arrives, an operation completes or at a fixed time, say at 6PM. We call this an *evalWhen* component of an SLO. Once the *evalWhen* event arrives, a set of samples of *measuredItem* are obtained applying a sampling function. The *evalOn* component determines how this sample is computed. The sample set is a constrained set of measured data that is constrained by the *evalOn* component. Examples of *evalOn* components may be a number or a time period, e.g. the 5 longest running transactions, or all the samples for last 24 hours. A function (*evalFunc*) is thereafter applied on the sample set so obtained. An example of *evalFunc* is the average response time function < 5 ms. The *evalFunc* must be a mathematical function that is expressible in terms of its inputs and logic. The following grammar shows a portion of this formalization. The *evalAction* specifies the actions after the evaluation is done.

```
SLA  = dateconstraint SLO*
Dateconstraint = startdate enddate nextevaldate
SLO = daytimeconstraint clause*
Daytimeconstraint = Day* time
Clause = measuredItem evalWhen evalOn evalFunc evalAction
MeasuredItem = Item*
Item  = measuredAt constructType constructRef
```

Fig. 12 SLA definition

As an example, a clause like *At 6 PM the Average response time for the 5 longest running bookbuy transactions measured on the client side should be < 5 ms* can be broken up into a *measuredItem* (Item: bookbuy transaction, measuredAt:Consumer), *evalWhen* (at 6PM), *evalOn* function (set of 5 longest running transactions), the *evalFunc* (average response time < 5 ms) and *evalAction* (Notify administrator). The complete set of examples of how complex SLAs can be represented in it are presented in [21].

References

[1] G. Kar, A. Keller, S. Calo, "Managing Application Services over Service Provider Networks: Architecture and Dependency Analysis," *Proceedings of 7th IEEE/IFIP Network Operations and Management Symposium (NOMS),* Honolulu, Hawai, USA, April 2000.

[2] P. Bhoj, S. Singhal, S. Chutani, "SLA Management in Federated Environment," *Computer Networks*, Vol. 35, No. 1, pp. 5—24, 2001. Also *HP Labs Technical Report* HPL-1998-203, 1998.

[3] U. Black, *ISDN and SS7 Architecture for Digital Signaling Networks,* Upper Saddle River, New Jersey, USA: Prentice Hall PTR, 1997.

[4] H. Sinnreich, A. B. Johnston, *Internet Communications Using SIP*, John Wiley & Sons, 2001.

[5] T. Yoshimura, Y. Yonemoto, T. Ohya, M. Etoh, and S. Wee, "Mobile Streaming Media CDN Enabled by Dynamic SMIL," *Eleventh International World Wide Web Conference (WWW11)*, May 7-11, 2002, Honolulu, Hawaii, USA.

[6] Flamenco Networks URL: http://www.flamenconetworks.com

[7] Kenamea URL: http://www.kenamea.com

[8] Talking Blocks URL: http://www.talkingblocks.com/

[9] B. Lhereux, "Web Services Networks Secure a New Technology," *Gartner Research Note* SPA-17-0627, July 2002.

[10] A. van Moorsel, "Ten-Step Survival Guide for the Emerging Business Web," to be published in *Lecture Notes in Computer Science: Web Services, e-Business and the Semantic Web: Foundations, Models, Architectures, Engineering and Applications*, Springer-Verlag, 2002, also HP Labs Technical Report HPL-2002-203, July 2002.

[11] A. Sahai, S. Graupner, W. Kim, "The Unfolding of the Web Services Paradigm," to be published in *Internet Encyclopedia*, J. Wiley, also *HP Labs Technical Report* HPL-2002-130, May 2002.

[12] R. Sturm, W. Morris, M. Jander, *Foundations of Service Level Management*, Sams, 2000.

[13] A. Keller, G. Kar, H. Ludwig. A. Dan, J. Hellerstein, "Managing Dynamic Services: A Contract Based Approach to a Conceptual Architecture," *Proceedings IEEE/IFIP Network Operations and Management Symposium (NOMS)*, pp.513—528, Firenze, Italy, Apr. 2002.

[14] A. Sahai, V. Machiraju, J. Ouyang, K. Wurster, "Message Tracking in SOAP-based Web Services," *Proceedings IEEE/IFIP Network Operations and Management Symposium (NOMS)*, Firenze, Italy, April 2002, also *HP Labs Technical Report*, HPL-2001-199, 2001.

[15] M. Sayal, V. Machiraju, A. Sahai, A. van Moorsel, "Correlators for Monitoring Web Services and Business Processes," to be published as *HP Labs Technical Report*, August 2002 (available from the authors).

[16] *EAI Journal*, Business Activity Monitoring Cover Issue, Vol. 4, Nr. 7, July 2002.

[17] S. Frølund, M. Jain, J. Pruyne, "SoLOMon: Monitoring End-User Service Levels," *Integrated Network Management VI—Distributed Management for the Networked Millenium, Proceedings of the 6th IFIP/IEEE International Symposium on Integrated Network Management* (IM), M. Sloman, S. Mazumdar, E. Lupu (Editors), pp. 261—274, IEEE Computer Society Press, Boston, MA, USA, May 1999.

[18] D. Keim, M. Hao, J. Ladisch, M. Hsu, U. Dayal, "Pixel Bar Chart:A Technique for Visualizing Large Multi-Attribute Data Sets Without Aggregation," *IEEE Information Visualization*, 2001, also *HP Labs Technical Report* HPL-2001-92, 2001.

[19] *OMI White Paper*, http://www.openview.hp.com/products/smart_plug-ins/tech_whitepaper/spi_webmethods_omi_twp_jun02.pdf

[20] A. Sahai, V. Machiraju, "A Data Model Based on Service and Process Abstractions for Management of Systems," *HP Labs Technical Report*, HPL-2002-190, July 2002.

[21] A. Sahai, A. Durante, V. Machiraju, "Towards Automated SLA Management," *HP Labs Technical Report,* HPL-2001-310, 2001.