Semantic Web Services

Web Service Execution Environment (WSMX)

© Copyright 2010: Dieter Fensel and Srdjan Komazec

Outline

- Motivation
- Technical solution
  - Overview
  - Introduction
  - Relation to the WSMO and WSML
  - Design principles
  - Lifecycles
  - Conceptual Model
    - Architecture
    - Components
    - System entry points
  - Behavioral Model
    - Execution semantics
- Illustration by larger example
- Extensions
- Summary
- References

Motivation

Where are we?

<table>
<thead>
<tr>
<th>#</th>
<th>Title</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Introduction</td>
</tr>
<tr>
<td>2</td>
<td>Web Science</td>
</tr>
<tr>
<td>3</td>
<td>Service Science</td>
</tr>
<tr>
<td>4</td>
<td>Web Services</td>
</tr>
<tr>
<td>5</td>
<td>Web2.0 Services</td>
</tr>
<tr>
<td>6</td>
<td>Semantic Web</td>
</tr>
<tr>
<td>7</td>
<td>Web Service Modeling Ontology (WSMO)</td>
</tr>
<tr>
<td>8</td>
<td>Web Service Modeling Language (WSML)</td>
</tr>
<tr>
<td>9</td>
<td>Web Service Execution Environment (WSMX)</td>
</tr>
<tr>
<td>10</td>
<td>OWL-S and others</td>
</tr>
<tr>
<td>11</td>
<td>Light-weight Annotations</td>
</tr>
<tr>
<td>12</td>
<td>Applications</td>
</tr>
<tr>
<td>13</td>
<td>Mobile Services</td>
</tr>
</tbody>
</table>

4/6/10
Motivation

How SESA emerged?

- Service Orientation in information systems
  - Service-Oriented Computing (SOC)
    - Services as the fundamental elements for the development of rapid, low-cost, and easily integrable enterprise applications,
    - Service-Oriented Architecture (SOA)
      - SOC can be abstractly implemented by SOA,
      - From function to objects to service,
      - SOA requirements: loose coupling, implementation neutrality, flexible configuration, long lifetime, granularity and teams.
- Existing technologies and SOA solutions are:
  - ... difficult to scale without a proper degree of automation,
  - ... partial solution to interoperability.
- Semantically Enabled Service Oriented Architecture (SESA) represents SOA empowered by adding semantics as a means to deal with heterogeneity and mechanization of service usage.

Motivation

What is SESA?

- Application of SESA offers a scalable integration, more adaptive to changes
  - service offerings and required capabilities are described by semantically rich and formal service models,
  - exchanged data is also semantically described, and
  - reasoning provides total or partial automation of tasks.
- A SESA implementation should build a layer on top of the existing technologies (e.g. Web Services).

Web Service Execution Environment (WSMX) is an implementation of SESA.

Motivation

SESA Architecture

Motivation

WSMX Goals

- Middleware for Semantic Web Services
  - Allows service providers to focus on their business,
- Reference implementation for WSMO
  - Eat our own cake,
- Environment for goal based discovery and invocation
  - Run-time binding of service requesters and providers,
- Provide a flexible Service Oriented Architecture
  - Add, update, remove components at run-time as needed,
- Keep open-source to encourage participation
  - Developers are free to use in their own code, and
- Define formal execution semantics
  - Unambiguous model of system behaviour.
WSMX is...
- ... is comprehensive software framework for runtime binding of service requesters and service providers,
- ... interprets service requester’s goal to
  - discover matching services,
  - select (if desired) the service that best fits,
  - provide data/process mediation (if required), and
  - make the service invocation,
- ... is reference implementation for WSMO,
- ... has a formal execution semantics, and
- ... is service oriented, event-based and has pluggable architecture
  - Open source implementation available through Source Forge,
  - based on microkernel design using technologies such as JMX.

Overview
Relation to WSMO and WSML
- Conceptual Model & Axiomatization for SWS
- Formal Language for WSMO
- Execution Environment for WSMO
- OASIS TC

Overview
Design principles
- Service-oriented principle
  - Service reusability, loose coupling, abstraction, composability, autonomy, discoverability,
- Semantic Principle
  - Rich and formal description of information and behavioral models enabling automation of certain tasks by means of logical reasoning,
- Problem-solving principle
  - Goal-based discovery and invocation of services, and
- Distributed principle
  - Executing process across a number of components/services over the network, thus promoting scalability and quality of process.
WSMX aims to support the complete Semantic Web Service lifecycle:

1) Discovery - determines usable services for a request,
2) Composition - combine services to achieve a goal,
3) Selection - chooses most appropriate service among the available ones,
4) Mediation - solves mismatches (data, protocol, process) hampering interoperability,
5) Choreography - interactions and processes between the service providers and clients,
6) Grounding - lifting and lowering between the semantic and syntactic data representations, and
7) Invocation - invokes Web service following programmatic conventions.

Technical Solution

Conceptual Model - Architecture

WSMX architectural components are divided into three layers

- **Execution Management layer**
  - Responsible for managing all other components and interactions between them.
  - Components are orchestrated in the form of Execution Semantics (explained in upcoming slides).

- **Broker layer**
  - Represented by a number of broker services each dedicated to accomplish different tasks in SWS lifecycle.

- **Base layer**
  - Provides support to various upper level components by providing storage services, as well as parsing and reasoning.

WSMX provides different entry points in order to enable users to consume the offered functionality.
Technical Solution
Conceptual Model – WSMX Components
Core Management

• Core Management is a kernel of WSMX with the following objectives
  – It realizes the overall operational semantics in order to achieve the functional
    semantics of its client-side interface
  – It orchestrates the functionality of the middleware components into a coherent process
    in an orderly and consistent fashion called Execution Semantics (see later)
  – It ensures the proper inter-component communication through publishing and
    subscribing to the data as sets of triples over triple space or directly in the
    synchronous communication manner

Conceptual Model – WSMX Components
Core Management

Technical Solution
Conceptual Model – WSMX Components
Communication Manager and Invoker

• Responsible for interaction with services and entities that are external to
  WSMX.
• Should be open to support as many transport and messaging protocols
  as possible (transparently to WSMX).
• WSMX uses
  – The SOAP implementation from Apache AXIS, and
• Both RPC and Document style invocations possible
Technical Solution
Conceptual Model – WSMX Components
Grounding

WSMX Components
Grounding

• WSMO service descriptions are grounded to WSDL by the means of XSLT lifting and lowering


Technical Solution
Conceptual Model – WSMX Components
Discovery
• Responsible for finding appropriate Web Services capable of fulfilling a goal
• Different techniques available
  – trade-off: ease-of-provision vs. accuracy
  – resource descriptions & matchmaking algorithms

Key Word Matching
- match natural language key words in resource descriptions,

Controlled Vocabulary
- ontology-based key word matching, and

Semantic Matchmaking
- what Semantic Web Services are at.

• Allows for a fast filtering and ranking of the huge number of available services rather quickly.
• Nonfunctional properties from the Dublin Core namespace (e.g. dc:description) are candidates for indexing and querying.
• Dictionaries of synonyms (WordNet) can be used to discover more services.

Exact Match: \( G, WS, O, M \vdash V x. (G(x) \leftrightarrow WS(x)) \)
PlugIn Match: \( G, WS, O, M \vdash V x. (G(x) \Rightarrow WS(x)) \)
Subsumption Match: \( G, WS, O, M \vdash V x. (G(x) \subseteq WS(x)) \)
Intersection Match: \( G, WS, O, M \vdash \exists x. (G(x) \land WS(x)) \)
Non Match: \( G, WS, O, M \vdash \neg \exists x. (G(x) \land WS(x)) \)

Conceptual Model – WSMX Components
Discovery – Key Word Matching

Conceptual Model – WSMX Components
Discovery – Simple Semantic Descriptions

Conceptual Model – WSMX Components
Discovery – Simple Semantic Descriptions - Example
• Exact match
  (1) Goal.TripAustria ⊑ Service.DEBB
  (2) Goal.Trip.V72DE ⊑ Service.VTA
  (3) Goal.Trip.BK2FRA ⊑ Service.VTA
  (4) Goal.TripsAustria ⊑ Service.VTA
  (5) Goal.Trip.BK2S2G ⊑ Service.VTA
  (6) Goal.Trip.BK2S2G ⊑ Service.DEBB
  (7) Service.VTA ⊑ Goal.TripEurope
  (8) Service_DB ⊑ Goal.TripEurope
  (9) Service.DEBB ⊑ Goal.TripEurope
  (10) Service_DBNight ⊑ Goal.TripEurope

• Plug-in match

• Subsumption match

• Intersection match
  (1) ¬(Goal.Trip.BK2FRA ⊑ Service_DBNight)

Technical Solution

Conceptual Model – WSMX Components

Ranking and Selection

• One service which best satisfies the user preferences is selected from the candidate services returned by the service discovery.
• Selection  
  – determines best candidate out of discovered WS,
• Ranking  
  – determines a priority list of discovered WS.
• The process is run after ‘functional’ discovery
• Criteria:  
  – Quality of Service (security, robustness, availability),
  – Context (regional, business / social communities),
  – Preferences and policies,
  – Financial criteria,
  – …

Conceptual Model – WSMX Components

Ranking and Selection

• Ontologies for specifying QoS (http://www.wsmo.org/ontologies/nfp)  
  – set of 17 ontologies (i.e. locative, temporal, availability, price, trust, security, etc.)  
  – provides the terminology needed to specify QoS aspects of services
• The data required for ranking and selection can be attached through the non-functional aspects of SWS

• The example defines the value of price for the Web Service consumption as a WSML logical expression which needs to be evaluated.
  – If the client is older than 60 or less than 100 years old the predefined absolute price is taken which must be less or equal to 10.
Service Ranking
- Based on order theory which defines notions of total order, partial order and ranking.
- WSMX uses Multicriteria ranking – considers multiple nonfunctional properties.

Service Selection
- User specifies selection criteria in terms of nonfunctional properties such as SLA, QoS, etc. Those can be obtained from goal description.
- Service defines nonfunctional property values are directly provided or computed/collated by a monitoring tool.

<table>
<thead>
<tr>
<th>Conceptual Model – WSMX Components</th>
<th>Conceptual Model – WSMX Components</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ranking and Selection</td>
<td>Ranking and Selection – Example</td>
</tr>
</tbody>
</table>

Technical Solution
Conceptual Model – WSMX Components
Data Mediation

- Ontology-to-ontology mediation
- A set of mapping rules are defined and then executed
  - Ontology Mapping Language
- Initially rules are defined semi-automatic
- Create for each source instance the target instance(s)
Design-time

- **Inputs**
  - Source Ontology and Target Ontology
  - Features
  - Graphical interface
  - Set of mechanisms towards semi-automatic creation of mappings
  - Capturing the semantic relationships identified in the process
  - Storing these mappings in a persistent storage
- **Output**
  - Abstract representation of the mappings

Run-time

- **Main Mediation Scenario:** Instance Transformation
- **Inputs**
  - Mediated data
  - Source ontology instances
  - Features
  - Completely automatic process
  - Grounding of the abstract mappings to a concrete language
  - Uses reasoner to evaluate the mapping rules
- **Outputs**
  - Mediation strategy
  - Source strategy
  - Destination strategy

Conceptual Model – WSMX Components

Data Mediation – Example

<Alignment>
<dc:identifier rdf:resource="m2"/>
<onto1>
<formalism name="WSML" uri="http://www.wsmo.org/wsml/"></formalism>
<uri>http://deri.org/iswc2005tutorial/ontologies/travel1#travel1</uri>
</onto1>
<onto2>
<formalism name="WSML" uri="http://www.wsmo.org/wsml/"></formalism>
<uri>http://deri.org/iswc2005tutorial/ontologies/travel2#travel2</uri>
</onto2>
<map>
  ...
    <entity1>
      <Class rdf:about="http://deri.org/iswc2005tutorial/ontologies/travel1#ticket"></Class>
    </entity1>
    <entity2>
      <Class rdf:about="http://deri.org/iswc2005tutorial/ontologies/travel2#travelVoucher"></Class>
    </entity2>
    <measure>1.0</measure>
    <relation>ClassMapping</relation>
  </Cell>
  ...
</map>
</Alignment>
Requester and provider have their own communication patterns

Only if the two match precisely, a direct communication may take place

At design time equivalences between the choreographies’ conceptual descriptions is determined and stored as set of rules

The Process Mediator provides the means for runtime analyses of two choreography instances and uses mediators to compensate possible mismatches

*Not implemented yet!*
• Not a priori compatible behavior interfaces for communication & information interchange
• Partially resolvable by “process mediation patterns”

Technical Solution

Conceptual Model – WSMX Components
Choreography

• Requester and provider have their own observable communication patterns
  – Choreography part of WSMO
• Choreography instances are loaded for the requester and provider
  – Both requester and provider have their own WSMO descriptions
• Abstract State Machines (ASM)-based Choreography Engine
  – Evaluation of transition rules
    – prepares the available data
    – Sends data to the Process Mediator
  – Filters, changes or replaces data
  – Receives data from PM and forwards it to the Communication manager
  – data to be finally sent to the communication partner
The model used to express typical characteristics of a choreography must be compliant to the following features:

- Abstract – hides away any details regarding the underlying message exchange protocols and message formats,
- State-based – the interactions are described in the form of updates on a state of the choreography,
- Expressive – allows describing features such as sequences of message interactions and branching, and
- Ontology Reliance – ontologies are used as the underlying data model for message exchanges.

The abstract state model chosen to address the requirements is inspired by the Abstract State Machines (ASM) methodology.

Transition rules define actual behavior of the choreography.

The rules can take the form of

- if Condition then Rules endIf – if the condition holds executes the updates,
- forall Variables with Condition do Rules endForall – simultaneous execution of updates for each binding of a variable satisfying a given condition,
- choose Variables with Condition do Rules endChoose – executes an update with an arbitrary binding of a variable chosen among those satisfying the selection condition.

Condition (guard) is an arbitrary logical expression as defined by WSML.

Rules may take the form of Updates, whose execution to be understood as changing (or defining, if there was none) instances in an ontology:

- add – adds new instances in the state ontology,
- delete – deletes instances from the state ontology, and
- update – updates instances in the state ontology.
Technical Solution
Conceptual Model – WSMX Components
Resource Manager

- Stores internal memory model to a data store
- Decouples storage mechanism from the rest of WSMX
- Data model is compliant to WSMO API
- Independent of any specific data store implementation i.e. database and storage mechanism
- Maintains six repositories to store
  - WSMO top level entities, i.e.
    - Goals,
    - Web Service descriptions,
    - Mediators,
    - Strategies,
    - Event data and intermediate messages
    - WSDL descriptions

Conceptual Model – WSMX Components
Reasoners

- Different components within WSMX necessitate efficient and different reasoning functionality:
  - Discovery:
    - Simple ontological reasoning and query answering as well as logical entailment between preconditions and postconditions of SWS and Goals
    - Both description logic-based and logic programming-based reasoning is required
  - Selection:
    - Evaluation of the logical rules that are used to model the non-functional properties of services
    - Logic programming-based reasoning is required
  - Data mediation:
    - Ontology mapping rules, source instances and source and target schema information are loaded into the reasoning space where rules are evaluated in order to produce target instances
    - Logic programming-based reasoning is required
  - Process Mediation:
    - Reasoning is used to check whether messages are expected at the certain phase of the communication
    - Evaluation of transition rules is required
  - Different reasoning functionality is provided to WSMX through WSML2Reasoning framework.

Technical Solution
Conceptual Model – WSMX Components
Reasoners
• Represents a generic, flexible architecture for reasoning with the different variants of the WSML family.

• Instead of implementing new reasoners it uses existing reasoner implementations for WSML through a wrapper that:
  1) maps WSML expressions into a common (shared) knowledge representation format (different formats for the rule-based and description logic based WSML variants), and
  2) maps the expressions via specific adapters into the appropriate syntaxes of concrete reasoning engines.

• People can use their specific existing reasoner of choice and can exploit already developed, proven and tested reasoning systems.

---

**Conceptual Model – WSMX Components**

**Reasoners - WSML2Reasoner Framework**

- **Reasoning with rule-Based WSML**
  - Covers WSML-Core, WSML-Flight, and WSML-Rule
  - Supported through semantics-preserving syntactic transformation of WSML ontologies to Datalog programs with (in)equity and integrity constraints.
    - Axiomatization – conversion of conceptual elements into appropriate axioms,
    - Normalization – reduction of complexity by bringing the expressions closer to the simple syntactic form of literals in Datalog rules
    - Lloyd-Taylor transformation – relaxing of the remaining complex WSML logical expressions in order to produce simple rules (single head and conjunctive – possibly negated – body literals).
    - Datalog rule generation – All WSML logical expressions are transformed into a Datalog program

- **Supported reasoning tasks**
  - Query answering, ontology consistency, entailment, retrieval

---

**Conceptual Model – WSMX Components**

**Reasoners - WSML2Reasoner Framework**

- **Reasoning WSML-DL**
  - Supported through semantics-preserving syntactic transformation of WSML-DL ontologies to OWL-DL ontologies.
    - Relations to attributes – replacing relations, subrelations, and relation instances by attributes and axioms,
    - Axiomatization – conversion of conceptual elements into appropriate axioms,
    - Implication reduction rules – replacing equivalences and right implications in logical expressions by left implications,
    - Inverse implication reduction rules – replace conjunctions on the left side and disjunctions on the right side of an inverse implication by left implications,
    - Molecular decomposition rules – replace complex molecules inside logical expressions by conjunctions of simple ones, and
    - OWL API transformation – each logical expression is translated into the corresponding OWL descriptions

- **Supported reasoning tasks**
  - Knowledge base consistency, concept satisfiability, concept subsumption, instance checking, realization, instance retrieval
Technical Solution
Conceptual Model – Entry Points

- Represent input ports to which messages can be sent for initiating specific execution semantics.
- Published as SOAP Endpoints
  - `getWebServices(WSMLDocument)`: Web Services
    - A service requester wishes to discover a list of SWS fulfilling its requirements provided by as a goal description using WSML.
    - A set of WSML Web Service descriptions whose capability matches the goal is returned.
  - `invokeWebService(WSMLDocument, Context)`: Context
    - Used to invoke already known Semantic Web Service by relying on data provided in the form of WSML, scenario, and conversation context.
  - `achieveGoal(WSMLDocument)`: Context
    - A service requester wishes to use WSML for all aspects of goal-based service invocation (discovery, mediation, invocation) by providing both goal and data in the single WSML document.
    - Processing of the message is identified by the conversation context.

Technical Solution
Behavioral Model

Behavioral Model
Execution Semantics

- Execution Semantics is a formal description of the operational behavior of the system in terms of computational steps.
- The benefits of having behavioral models are in
  - Greater flexibility in SESA implementations,
  - Foundations for model testing,
  - Executable representation, and
  - Improved model understanding among humans.
- Mandatory execution semantics
  - Goal-Based Web Service Discovery
  - Web Service Invocation
  - Goal-Based Service Execution
Behavioral Model
Execution Semantics – Goal-based discovery

• Input
  Service requester wishes to discover a list of SWS fulfilling its requirements provides them as a goal description using WSML.

• Output
  A set of WSML Web Service description whose capability matches the goal is returned.

• Process
  1. Message containing WSML goal is received and passed to the communication manager, which takes care of message persistence.
  2. The goal is sent to discovery activity which evaluates whether a WSML Web service from the repository is capable of fulfilling the goal. It may need data mediation as its internal step.
  3. The selection activity may be triggered to determine which of the discovered services to return to the service requester.
  4. WSML Web service description is returned in a form of message to the requester by relying on the Communication Manager facilities.

Behavioral Model
Execution Semantics – Web Service Invocation

• Input
  As input the requester is providing WSML content (Web service + data) and conversation context (if known!!! The first message is sent without it, but it is returned as output of the invocation).

• Output
  Results of the Web Service invocation (and possibly conversational context)

• Process
  1. Communication Manager unpacks the content and retrieves Web service and data.
  2. Data mediation may be required in cases where heterogeneities exist (this step can internally use reasoner).
  3. Process mediation has all the required data to commence (matching the message patterns and data types offered by SWS and required by the goal). This is a process of aligning of input data instances so that they can be understood by the Choreography.
  4. The choreography activity is commencing (determines which messages to send on the basis of choreography descriptions – ASMs).
  5. Grounding is taking place in order to transform those instances to the format required by the involved service.
  6. Invocation takes place.
• Service requester wishes to employ the complete SWS lifecycle, i.e., all aspects of goal-based service invocation (discovery, mediation, invocation).
  • Input:
    - Requester provides both the goal and the input data descriptions in a single WSML document.
  • Output:
    - Results of the Web Service invocation (if any).

• Operation:
  1. Execute the Goal-based Web Service discovery execution semantics.
  2. In case of more than one result run the selection process in order to determine the Web Service which is the most appropriate for the execution (based on non-functional properties and employed selection mechanisms).
  3. Execute the Web Service Invocation execution semantics.

Illustration by a Larger Example

• The goal is to discover a suitable solution for the transportation of a package with defined size and weight.
• Candidate Web Services have different constraints regarding the transportation destinations, package size and weight acceptance, as well as pricing schemas.
• For more information visit:
I want to have my package shipped from CA, USA to Tunis, Africa size (7/6/4), weight 1 lbs, the cheaper the better.
Discovery is employed in order to find suitable Web Service
Discovery consults appropriate ontologies and Web Service
descriptions
Web Service may be invoked in order to discover service availability

Africa ($85.03/lb) Max 50 lbs. Price = $85.03

Ships only to US ($10/1.5 lb).
Cannot be used for Africa.
PriceReq
Price ($65.03)

Requester and provider choreographies are instantiated and processed
Invocation of Web Service occurs
Possible Extensions

Discovery

- Discovery component has undergone a set of smaller improvements:
  - Instead of full Semantic Web Service descriptions, results are returned in the form of WG mediators.
  - Caching of the discovery results, i.e., WG mediators.
- WG mediators are allowing us to:
  - Connect Goal descriptions with the Semantic Web Service descriptions.
  - Attach discovery (e.g., type of match) and ranking results (e.g., fitness of the service).
- Caching of the results enables reductions of the computational time required for discovery when the same Goal is issues multiple times over the same set of SWS descriptions.
WSMX can be viewed as a vibrant events producer – events related to the broker service execution, – events related to the external Web Service invocations, – events related to the progress of processed execution semantics.

The events represent a rich source of knowledge which can facilitate profiling, optimization and reliability of the environment.

The monitoring and complex event processing subsystem exhibits following behavior – WSMX components have been instrumented to emit appropriate semantically-described events in regard to the computational process progress, – RDF-based CEP engine detects occurrences of specified event patterns and executes the appropriate actions defined for the detected event (e.g. update of the aggregated statistical measures, notification of the administrator about the successive operation failures).

Notification Broker enables asynchronous communication within WSMX

The broker provides following functionality – registering and storing user goal-descriptions and related results, and – notifying subscribed users of the related results.

The broker currently supports the subscription to the goal-based Web Service discovery through a new entry point which fires the execution semantics responsible for – checking for the existing (and not expired) user subscriptions, – executing for each subscribed goal the discovery process, and – once new services have been matched with respect to the previous subscriptions these newly discovered services are notified to the users.

The notification can employ two communication channels – Email notification, and – Web Service invocation.

Orchestration Engine features are – support of the dataflow in a manner consistent with WSMO/L, and – Adoption of the explicit concept of performance, and introduction of a new type of mediator to mediate between performances.

Orchestration Engine solves following issues – mediation in any connection, including dataflow, between heterogeneous components; – extraction and aggregation in the consumption and production of messages according to the WSMO grounding approach; – execution of service discovery and service invocation within composed services.

Summary
Summary

- Clear separation of SWS, SESA, SEE and WSMX notions.
- WSMX is reference implementation of WSMO.
- WSMX contributes to solving scalability issues by automating various service-related tasks.
- WSMX consists of various components dedicated to solving particular steps in the overall service life cycle orchestrated by Core Management.
- Execution semantics is a formal description of the operational behavior of the system in terms of computational steps.

References

- Mandatory reading
  - Chapters 4, 5, and 6.
- Optional reading
  - Charles Petrie, Tiziana Margaria, Holger Lausen, Michal Zaremba (Eds.), Semantic Web Services Challenge: Results from the First Year. Springer-Verlag, 2008.
- Online resources:
  - http://see.sti-innsbruck.at
  - http://sws-challenge.org

Next Lecture

<table>
<thead>
<tr>
<th>Title</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>3</td>
</tr>
<tr>
<td>4</td>
</tr>
<tr>
<td>5</td>
</tr>
<tr>
<td>6</td>
</tr>
<tr>
<td>7</td>
</tr>
<tr>
<td>8</td>
</tr>
<tr>
<td>9</td>
</tr>
<tr>
<td>10</td>
</tr>
<tr>
<td>11</td>
</tr>
<tr>
<td>12</td>
</tr>
<tr>
<td>13</td>
</tr>
</tbody>
</table>
Questions?