Intelligent Service Integration – overview of work at LIA-EPFL

Ion Constantinescu, Boi Faltings, Steve Willmott

Artificial Intelligence Laboratory, Swiss Federal Institute of Technology, IN (Ecublens), CH-1015 Lausanne, Switzerland.

{ion.constantinescu, boi.faltings, steve.willmott}@epfl.ch

Abstract

Current web infrastructure is oriented on human-machine interactions. Developments for next generation systems (such as Web Services and ebXML) however aim to allow for automated interactions between arbitrary systems / services. This is achieved by supplying high-level descriptions of service capabilities and enabling information systems to dynamically discover and access each other.

This paper takes into account 5 well-known behavioral description languages (PDDL, DAML-S, WSFL, ebXML and ConGolog), tries to propose an abstract representation behavior representation and gives a mapping between it and the existing formalisms.

Keywords: service integration, interaction protocols, process definition, planning, multiagent systems.

1 Introduction

Today, the largest human designed and controlled environment is the Internet. The Internet can be seen as a dynamic environment with a huge heterogeneous collection of infrastructures and services. There are many different types of architectures and several metaphors for using them but central to all is the concept of "service" providers of static (e.g. informational web pages) or active (e.g. news alerts service) resources. Active resources allow on their invocation for changes in the environment to be effected. As the number of services increases so does the need for service reuse and service composability.

The underlying problem that needs to be solved in order to realize the above vision is directly related with much of the research that has been carried out in the Agent and AI communities. Service integration can be seen as a complex coordination problem [20] (since services must work together over a period of time to achieve the initial user request) with a reasoning problem at its core (constructing a multi-agent plan to determine how services will work together).

In this paper we are concerned with the formalisms used to described services and in particular the *behavioral descriptions* which could be used to reason about composing services to form value added services. In particular we briefly introduce a number of existing description formalisms (Section 3) and subsequently argue for the development of an Abstract Behavior Representation (ABR) - Section 4. Section 5 briefly outlines an example ABR and Section 6 concludes the paper. We begin with a brief introduction to the challenges of service integration in open environments such as Agentcities.

2 Service Integration Challenges

The challenges related to automated service integration in open environments might be divided into four steps:[1]

- **Problem Identification**: Identifying and describing an integration problem (this may be simple goal setting or come about through an agent monitoring the environment for certain types of opportunities).
- **Team Formation (Discovery / Binding)**: discovery of appropriate services in the world to help solve the problem identified (usually via reference to their service descriptions). Also it can be the binding of an advertisement for the problem in the environment. Either of these involves communication relating to the problem
- **Plan formation (Reasoning)**: establishment of a plan of action to solve the problem identified
- **Joint Action (Execution)**: execution of the established plan in the environment by the systems forming part of the team.

A critical factor permeating all of these levels is the *description* of services in the environment.[2] These affect in particular team formation (discovering / binding relevant service information), plan formation (reasoning over description to devise a workable plan). In an open environment such as Agentcities or future Web Services environments this raises a number of challenges including the following:

- **Description:** how would systems operate if multiple description formalisms were used (which seems very likely to occur)? How can multiple formalisms be

composed (the DAML-S language for example leaves unspecified how pre-conditions might be expressed)?

- **Reasoning:** what is the impact of reasoning complexity for planning of each construct allowed in a particular description formalism? How can reasoning complexity be controlled?
- **Execution:** what flexibility is provided for description and management of runtime execution of resulting plans – in a multi-agent, hostile and dynamic environment?

3 Existing behavioral formalisms

We begin by briefly reviewing a number of existing behavioural description formalisms that appear particularly relevant to the problem of service integration. For a full description of those formalism see [5].

The Planning Domain Definition Language (PDDL) [10][9][11], was developed as a problem-specification language for the AIPS-98 planning competition. PDDL draws from existing formalisms like ADL, SIPE-2, Prodigy-4.0, UMCP, Unpop and UCPOP.

DAML-S - stands for DARPA Agent Markup Language (DAML) Semantic Markup for Web Services [6][18]. One of the targets of DAML-S is to enable the automatic selection, composition and interoperation of Web services to perform some task, given a high-level description of an objective. Another objective is service monitoring that should allow users or agents to determine the state of long-running services and interact with their execution.

The Web Services Flow Language (WSFL) is an XML language for the description of the composition of Web Services (see [16]). It aims to specify interaction pattern either internal to an existing Service composed of other services or external describing interactions between composed services.

ebXML [3] is an XML based language with the objective of providing an open XML-based infrastructure enabling the global use of electronic business information in an interoperable, secure and consistent manner. The Business Process Specification Schema is an ebXML document which describes in detail how Trading Partners take on roles, relationships and responsibilities to facilitate interaction with other Trading Partners in shared collaborations. The interaction between roles takes place as a choreographed set of business transactions. Each business transaction is expressed as an exchange of electronic Business Documents.

ConGolog [12][14] is an extended version of the Golog (AlGOl in LOGic) [15] language. Golog was originally developed as a high level language for programming robots and software agents. ConGolog is based on a logical formalism, the situation calculus [17], and can model multi-agent processes, non-determinism and concurrency. Traditionally simulation was the big virtue of state based formalisms and reasoning over the formal properties of the model was the main advantage of predicative models. By doing a logical definition of an application domain the ConGolog language can support both simulation and verification.

	ABR	PDDL	DAML-S	WSFL	ebXML	ConGolog
Core Concept	ServiceDescr	Action	Process	Activity, ServiceProvider	BusinessActivity	Action
Recursivity	Composite ServiceDescr	1.0: Action	Composite Process	FlowModel ServiceProvider	CollaborationActivity Binary Collaboration	Procedure
End-level Concept	ServiceDescr	Action	Atomic Process	PortOperation, ServiceProvider	Transaction Activity	Action
Top-level Concept	ServiceDescr	Domain	Process Model	GlobalModel; ServiceProvider	Multiparty Collaboration	Domain dynamics + processes
Input	Input	Parameters	Input	Input	Requesting Flow	Predicative Parameters
Output	Output	Parameters	Conditional Output	Output	Response Flow	Not specified
Failure	Exception	Not explicit	Not Explicit	Failure	Failure State	Not specified
Precondition	Precondition	Precondition	Precondition	Transition, Join Conditions	Precondition, Begins When	Precondition
Precondition spec	Bool expr	Full FOL	Thing	XPath	String	Full FOL
Effect	Effect	Conditional Effect	Conditional Effect	Exit Condition	Effect, Success / Failure Guard, Ends When	Conditional Effect
Effect	Conditional	Universal quant,	Thing	XPath	String	Full FOL

Specification	assignment	fluent assignments.				
Control Constructs	Nondeterministic, concurrent, If-then-else, While-do / Do-while	1.0: Choice, Series, Foreach, Forsome, Parallel	Split, Split-join, Sequence, Choice, TestCondition, Unordered, Iterate, If-Then-Else, Repeat-While, Repeat-Until	If, Do-loop	Start, Success, Failure, Fork, Join, Test Guard	Sequence, Choice, Iteration, If-Then-Else, While-do, Concurrent. Execution. Concurrent Execution with priorities. Concurrent Iteration, Interrupt, Procedure call
Time	Start, end, duration	Start, End, Duration	Start, End, During, Timeout	Duration, Retry	timeToPerform, timeToAckReceipt timeToAckAccept	Not explicit
Domain Constraints	Ontologies	1.0: axioms + safety constraints	Ontologies	No	No	Frame axioms, Trans, Final
Pred / numeric	Yes	Yes	No	No	No	No
Exogenous Actions	Yes- external	Level 5 only	No	No	No	Yes
Execution Control	Control Operations	Not specified	Stub Process Control Model	Control Operations	Not specified	Not specified
Execution Monitoring	Lifecycle status, contact info	Not specified	Lifecycle status	Lifecycle status, Contact info	Lifecycle status	Not specified

Table 1: A mapping between existing formalisms and the proposed ABR

In Table 1 we try to capture the main features of the above languages and do a mapping between them and our formalism proposed next.

Note: for some of the formalisms the recursive case is slightly different from the core case:

- DAML-S - specifies Computed IOPEs instead of plain IOPEs
- WSFL - FlowModels don't specify preconditions and exit conditions
- ConGolog procedures don't specify preconditions and effects

The definition of our formalism is structured in two sections: a planning section which has as objective to describe the model of a given domain and an execution section which defines a number of features useful for supporting the execution of the domain model.

4 Abstract Behavior Representations

Having reviewed a number of existing formalisms we argue that it would be of considerable value to develop an abstract behavioral description formalism.[3] In particular this could:

- Act as a partial "interlingua" between existing and future description formalism – allowing at least partial mappings between different formalisms.
- Provide common terms of reference for discussions on service descriptions at a level above syntax and interfaces currently possible.
- Enable initial characterization an consideration of the complexity and other properties of particular combinations of languages.[4]

5 An Example ABR

Clearly building (much less agreeing upon) such an ABR is a non-trivial task. We begin with a brief example that is based on concepts drawn from the 5 formalisms reviewed in Section 3.

5.1 Service Integration model

Our approach for Service Integration relies on a model containing mainly two steps: **problem identification** and **deployment**.

Deployment is the process by which a service gets actually used. Either from a service consumer perspective (e.g. user) that has some needs and requests some results or from a service producer perspective that makes available his capabilities for a given reason.

[3] Note that this idea has been mentioned in various discussion forums – please see acknowledgements section.

[4] Planning languages for example have seen a lot of work on problem complexity they generate and this should be leveraged in the service composition context.

Figure 1 describes some of the possible interactions between the three elements of our system and exterior entities such as users, sensors, binding repositories/matchmaking systems and actual service instances.

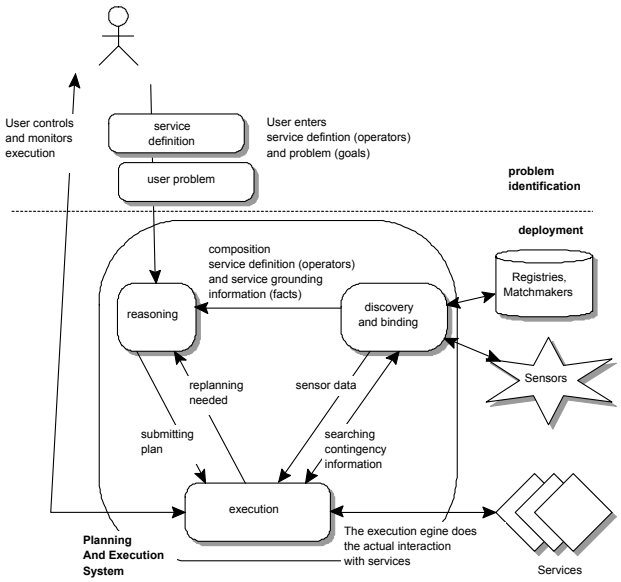

Figure 1: Model for Service Integration

5.2 Planning

In Figure 2 we describe visually the structure of our proposed formalism using the Unified Modeling Language (UML).

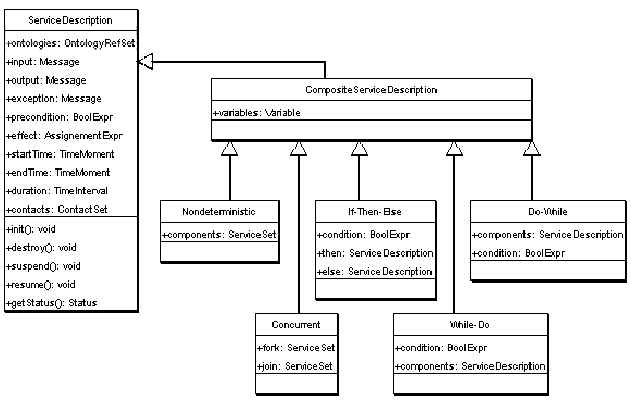

Figure 2: UML description of the proposed ABR

5.2.1 ServiceDescription

The core concept of the ABR is the ServiceDescription. A ServiceDescription is the most basic item of the ABR and each ABR model specifies at least one. A ServiceDescription defines also an invocation pattern for the underlying service that can be request/response, submit/response, notification or one-way message. A ServiceDescription has a number of attributes described below.

Ontologies: Each ServiceDescription refers to a number of ontologies that define the domains of parameters like input, output, exception, precondition and effect and a number of constraints between these parameters.

Input: Defines the information that the service needs to be passed upon invocation.

Output: Defines the information that the service produces upon successful invocation.

Exception: Defines the information that the service produces upon abrupt termination of the invocation (e.g. due to a failure).

Precondition: Is a boolean expression (see below) that defines the state of affairs required before the invocation of the service. For standalone ServiceDescriptions (not contained by a CompositeServiceDescription) the **precondition** expression can refer only to information specified in the **Input** parameter.

Effect: Defines the state of affairs generated by the invocation of the service. The effect is specified as a conjunction of assignments conditioned by boolean expressions (e.g. (and (when boolean expr1 assign 1) (when boolean expr2 assign 2) (always assign3)). "always" is equivalent with a condition with the boolean expression true. When defining standalone Services (not contained by a composite service) the **effect** boolean expression and assignments can refer only to information specified by the **output** or **exception** parameters.

Time: Since real-life systems need to be bounded by time constraints a service can specify a number of time parameters like Start, End and Duration. Still it is subject to discussion if this can be considered as a planning feature or has to be kept for the moment as an execution feature since reasoning over time domains can prove to be more difficult.

5.2.2 CompositeServiceDescription

The CompositeServiceDescription is used for introducing recursivity in our model. Also a CompositeServiceDescription serves as a "scope" or "world" for the contained services.

Variables: A composite service can explicitly define a number of variables over which the contained Services can evaluate preconditions, effect value changes and which they can use in the input, output or exception parameters. Also the input, output and exception parameters of the composite service define implicit variables. The preconditions for the composite service can be considered as initial values for those variables.

A composite service is an abstract description that is extended by a number of concrete specifications below. The most important is the Nondeterministic description since all others can be seen as syntactic sugar that can be then expanded to it.

5.2.3 Nondeterministic

The Nondeterministic serivce only defines a set of contained "components". The choice of the components is not prescribed and is limited by only the preconditions and effects of the components.

5.2.4 Concurrent

A concurrent service defines a concurrent execution of services. Two sets of services can be specified: a fork set and a join set. Services specified by the fork set are started in parallel. Then the concurrent service waits for services specified in the join set to complete and then returns.

5.2.5 If-Then-Else

A conditional construct uses a boolean expression condition to test which of two contained services to execute.

5.2.6 While-do / Do-while

Looping constructs use a boolean expression as a condition for repeatedly invoking a contained service. In the case of While-do the condition is evaluated before the invocation. In the case of the do-while the condition is evaluated after the service invocation.

5.2.7 Variables

Variables defined by a composite service designate the constructs that can appear in preconditions and effects of the contained Services. We assume that they are "internal" and discrete and are defined over enumerable domains. Internal in the sense that their value is modified only by the contained Services. Discrete in the sense that their value can be evaluated and changed only when a contained Service starts or ends. In other words their value cannot be accessed or changed by a Services while it runs (see [10][5]) for a discussion on how continuous variables can be converted to discrete). An enumerable domain is considered here to be a domain for which the size and the elements can be computed.

Predicative Variables: describe the truth-value of a literal grouping together a tuple of values.

Numeric Variables: describe a numeric state and can be integer or floating point.

Initial values: composite services can specify also initial values for explicitly defined predicative or numeric variables.

5.2.8 Boolean expressions

A boolean expression is a construct built using operators like not, and, or, iff, implies, ==, !=, <, >, <=, >= and implicit or explicit variables.

5.2.9 Assignment expressions

In the case of predicative variables the assignment can be either true or false. A true assignment can also be specified by the plain expression of the predicate (e.g. open(door1)) and a false assignment can be specified by as a plain expression of the predicated preceded by a not construct (e.g. not open(door1)).

In the case of numeric variables the value can be changed either by the direct assignment of a constant variable or by the change of the value with by a given constant. Supported changes are value increase/decrease and scale-up/scale-down.

5.2.10 Partial service descriptions, user problems and goals

There is a direct interdependency between the abstraction level of the problem specification, the reasoning support and the flexibility of the system.

As such some systems (e.g. intelligent agents) can usually handle high-level problem descriptions and be very flexible in terms of low-level choices. On the other hand this usually is expensive in terms of system design, as it requires highly trained professionals and in terms of reasoning support as it requires more powerful reasoning engines.

Other systems work using a lower level formalism (e.g. RPC oriented Web Services). Chaining together a number of such services using a programmatic formalism leads to smaller flexibility but it's easier to understand and develop and requires less powerful reasoning capabilities.

Our approach is to consider the gap between the two approaches as a continuum and allow different levels of completeness of behavior descriptions. In other words we allow for integration problems and solutions to be specified using the same formalism and be differentiated by different degrees of completeness.

For example for initiating a service integration process a user might specify an incomplete service description with the actual goals to be achieved as the **effect** attribute. Or the result of a planning reasoning over a CompositeServiceDescription might be a new more specific CompositeServiceDescription.

5.2.11 External Services and Variables

A number of existing formalisms (PDDL[10] Level 5, ConGolog [12][14]) introduce the concept of exogenous (external) actions. These are actions that are not in the control of the planning entity and can happen at random moments. From that we can easily derive the concept of external variables as variables modified or by such actions or provided to such actions. In our proposal we propose to use the "external" identifier for designating such variables or services. As it is not clear what kind of reasoning can be performed over them we submit this as a subject for more discussions.

5.3 Execution

For controlling the execution of a service we define a number of operations:

Init: Called in order to initialize a service instance.

Destroy: Called to signal that the execution of the service instance has to be aborted and resources have to be freed.

Suspend: Called for suspending the execution of a service instance.

Resume: Called for resuming the execution of a service instance.

For monitoring the execution we define the following features:

Contacts: An attribute listing contact information of persons actually responsible for the execution of the service and which can be contacted for providing execution support.

getStatus, subscribeMonitor, unsuscribeMonitor: for execution each service has an associated status reflecting lifecycle properties. The current state can be queried or it can be monitored by subscribing for notifications of status change.

6 Conclusion

In this paper we outline some of the challenges involved in enabling automated service integration in open environments and in particular discuss issues related to the behavioral description of services. We review a number of existing representations and argue that the development of an abstract behavior representation would be a useful step in supporting service integration in both Agentcities and other such environments. An example ABR is also given and is intended to:

- Encourage for discussion on practical requirements for behavioral service descriptions in the frame of the Agentcities network.
- Serve as an initial blueprint for the design of a system for automatic service integration.

7 Acknowledgements

The research described in this paper is partly supported by the EC project Agentcities.RTD (IST-2000-28385). The opinions expressed in this paper are those of the authors and are not necessarily those of the EU Agentcities.RTD partners. We would also like to state that there has been some discussion on abstract service descriptions in the context Agentcities Service Description and Web Services working groups as well as the FIPA Web Services Activity and we intend this paper to be a contribution to those discussions.

References

[1]. *International Planning Competition*, http://www.dur.ac.uk/d.p.long/competition.html, 2002.

[2]. A. Ankolekar, M. Burstein, J. Hobbs, O. Lassila, D. Martin, D. McDermott, S. McIlraith, S. Narayanan, M. Paolucci, T. Payne, K. Sycara, and H. Zeng. *DAML-S (version 0.6) Walk-Through*, 2001.

[3]. Business Process Project Team. ebXML *Business Process Specification Schema Version 1.01*, May 2001.

[4]. World Wide Web Consortium. *XML Path Language (XPath) Version 1.0*, 1999.

[5]. I. Constantinescu and B. Faltings. *Behavior Description Formalisms for Service Integration:* Technical Report No TBD. Technical report, Artificial Intelligence Laboratory, Swiss Federal Institute of Technology, 2002, http://liawww.epfl.ch/.

[6]. DAML Services Coalition (alphabetically A. Ankolekar and M. Burstein and J. Hobbs and O. Lassila and D. Martin and S. McIlraith and S. Narayanan and M. Paolucci and T. Payne and K. Sycara and H. Zeng). *DAML-S: Semantic Markup for Web Services.* In Proceedings of International Semantic Web Working Symposium (SWWS), 2001.

[7]. DARPA Agent Markup Language Program. *Reference description of the DAML+OIL (March 2001) ontology markup language*, 2001.

[8]. R. Fikes and N. Nilsson. STRIPS: *A new Approach to the Application of Theorem Proving to Problem Solving.* In Artificial Intelligence, 1971.

[9]. M. Fox and D. Long. PDDL+: *An extension to PDDL2.1 for modeling planning domains with continuous time-dependent effects*, 2002.

[10]. M. Fox and D. Long. PDDL2.1: *An Extension to PDDL for Expressing Temporal Planning Domains*, 2002.

[11]. M. Ghallab, A. Howe, C. Knoblock, D. McDermott, A. Ram, M. Veloso, D. Weld, and D. Wilkins. *PDDL-The Planning Domain Definition Language*, 1998.

[12]. Giuseppe De Giacomo, Yves Lesperance, and Hector J. Levesque. *ConGolog, a concurrent programming language based on the situation calculus.* Artificial Intelligence, 121(1-2): 109-169, 2000.

[13]. O. Lassila and K. Swick. *Resource Description Framework (RDF) model and syntax specification.* Technical report, World Wide Web Consortium, 1999.

[14]. Yves Lesperance, Todd G. Kelley, John Mylopoulos, and Eric S. K. Yu. *Modeling dynamic domains with ConGolog.* In Conference on Advanced Information Systems Engineering, pages 365-380, 1999.

[15]. Hector J. Levesque, Raymond Reiter, Yves Lesperance, Fangzhen Lin, and Richard B. Scherl. *GOLOG: A logic programming language for dynamic domains.* Journal of Logic Programming, 31(1-3): 59-83, 1997.

[16]. F. Leymann. *Web Services Flow Language (WSFL 1.0)*, May 2001.

[17]. J. McCarthy and P. Hayes. *Some philosophical problems from the standpoint of artificial intelligence*. In Machine Intelligence, 1969.

[18]. S. McIlraith, T.C. Son, and H. Zeng. *Mobilizing the Semantic Web with DAML-Enabled Web Services.* In Proceedings Second Int'l Workshop Semantic Web (SemWeb'2001), 2001.

[19]. Sheila McIlraith and Tran Cao Son. Adapting *Golog for programming the semantic web.*

[20]. M. Wooldridge and N. R. Jennings. *Towards a Theory of Cooperative Problem Solving.* In Proceedings Workshop Modelling Autonomous Agents in a Multi Agent World (MAAMAW'94), pages 15-26. 1994.