

A request language for web-services based on planning and constraint satisfaction

M. Aiello¹, M. Papazoglou^{1,2}, J. Yang², M. Carman³, M. Pistore³, L. Serafini³,
and P. Traverso³

¹ DIT — University of Trento, Via Sommarive, 14, 38050 Trento, Italy
contact author, phone +39-0461-882055
aiellom@dit.unitn.it,

WWW home page: <http://www.dit.unitn.it/~aiellom>

² INFOLAB — University of Tilburg, PO Box 90153, NL-5000 LE Tilburg, NL
{mikep,jian}@kub.nl,

WWW home page: <http://infolab.kub.nl/people/{mikep,jian}>

³ ITC-IRST, Via Sommarive,18, 38050 Trento, Italy

{carman,pistore,serafini,traverso}@irst.itc.it,

WWW: <http://sra.itc.it/people/{carman,pistore,serafini,traverso}>

Abstract. One of the most challenging problems that web-service enabled e-marketplaces face is the lack of support for appropriate service request languages that retrieve and aggregate services relevant to a business problem. We present an architectural framework for web-service interaction based on planning and constraint satisfaction, and a web-service request language (WSRL) developed on the basis of this framework. This framework is capable of performing planning under uncertainty on the basis of refinement and revision as new service-related information is accumulated (via interaction with the user or UDDI) and as execution circumstances necessitate change.

1 Introduction

The current phase of the e-business revolution is driven by enterprises that look to B2B solutions to improve communications and provide a fast and efficient method of transacting with one another. E-marketplaces are the vehicles that provide the desired B2B functionality. An e-marketplace is an electronic trading community that brings multiple customers, suppliers, distributors and commerce service providers in any geographical location together to conduct business over the Internet to produce value for end-customers and for each other.

A vertical e-marketplace, e.g., semiconductors, chemicals, travel industry, aerospace, etc, provides value by efficiently managing interactions between buyers and sellers in a specific industry. A vertical e-marketplace identifies a shared business vocabulary, standard business processes and product categorisation tailored to a particular industry. Further, the e-marketplace establishes and uses the shared meaning of these products and business processes to facilitate trading partner integration needs. The business interactions between organisations that

make up a business process are standardised and formally described, e.g., modelled using UMM (UN/CEFACT Modeling Methodology N090), and published via e-business directories such as the UDDI. Vertical e-marketplaces provide to their members a unified view of sets of products and services and enable them to transact using diverse mechanisms, such as web-services, available in the e-marketplace. This allows companies to conduct electronic business, by invoking web-services, with all partners in a marketplace rather than just the ones with whom they have collaborative business agreements. Service offers are described in such a way that they allow automated discovery to take place and offer request matching on functional and non-functional service capabilities.

One of the most challenging problems that web-service enabled e-marketplaces face is the lack of support for appropriate service request languages that retrieve and aggregate services relevant to a business problem. To understand this we may use an application scenario in the business domain of e-travelling and in particular the specifications of the open travel agency [1]. OTA has specified a set of standard business processes for searching for availability and booking a reservation in the airline, hotel and car rental industry, as well as the purchase of travel insurance in conjunction with these services. OTA specifications use XML for structured data messages to be exchanged over the Internet. Business processes like these specify how agents in an e-travelling marketplace interact in order to satisfy a traveller's requests. Key capability for a service request language would be to satisfy a user request by transparently assembling service solutions from different serviced providers in the e-marketplace dynamically, by aggregating and composing services and by constructing an end-to-end holiday package comprising a number of optimised flight and accommodation choices.

In this paper, we concentrate on the use of planning mechanisms for use within the context of web-services. In particular, our research aims is to provide higher-level lightweight constructs for a service request language whose semantics are based on extended temporal languages used for model based planning and on linear constraint satisfaction. In that respect, we combine a goal language for expressing extended goals in non-deterministic domain [6] with a system-level planning language for interacting with composed web services [7].

The paper is organised as follows: in Section 2, we relate planning to web-services. In Section 3, we introduce a domain for e-travelling. In Section 4, we illustrate the proposed approach by defining a formal request language, its interpretation, execution model, and a possible implementation. We show an example of a request and its execution in Section 5.

2 Planning for web-services

The field of AI planning offers high potential for solving problems in the context of web-enabled applications by instilling planning and scheduling capabilities to distributed decision-making agents in order to manage resource-constrained domains [2]. Recently, different planning approaches have been proposed (see [3–5]), most of which focus on gathering information and on applying deterministic

planning. In our view AI planning provides a sound framework for developing a web-services request language and for checking the correctness of the plans that are generated by it. Our work concentrates on developing a service request language for e-marketplaces. This request language results in the generation of executable plans describing both the sequence of plan actions to be undertaken in order to satisfy a request and the necessary information and constraints essential to develop each planned action.

There are a number of requirements that web-service centred planning should satisfy. The plans should have an “open dynamic structure” and be situated in the context of execution. This implies that plans should deal with non-deterministic effects of the business domain and the set of potentially executable actions may change during the course of planning. Hence, plans should be amenable to refinement and revision as new information is accumulated (via interaction with the user or UDDI) and as execution circumstances necessitate change. For instance, once a travel plan has been generated and proposed to the user, the user may decide to change some of the parameters connected to an action, e.g., hotel booking, by dynamically reconfiguring the plan. As plans inevitably do not execute as expected there is the constant need to be able to identify critical constraints and decision trade-offs, and for evaluation of alternative options. In the planning parlance, this means that it should be possible to deal with interleaved plans, interactive plans, reactive plans that deal with exogenous events, e.g., information supplied by the UDDI, and contingency plans that deal with uncertain outcomes of non-deterministic actions. This need for dynamic planning capabilities is at direct odds with classical planning research that assumes complete knowledge of the conditions in which the plan will be executed, deterministic execution and actions with predictable outcomes and little space for incremental changes. In addition to these requirements, there is also the core requisite of being able to prove the correctness of the correspondence between a request (goal) and the plans generated by the request language as a response to this request.

To comply with the above requirements, we have chosen to: (1) to use a model based planner for non-deterministic domains, e.g., open travel, and a standard constraint satisfaction solver; and (2) work with a combination of an extended temporal language and a linear constraint satisfaction language for expressing the goals (requests). The service request language can express information both about the succession of activities and over the parameters of the planned actions; while, the plans generated by the request language are capable of dealing with non-determinism, interleaving and constraints resolution.

3 The domain of e-travelling

To describe the service-based interaction between a traveller, a travel agency and a tour operator, we use the business process specification found in [1]. This business process specifies the format of a user request and the replies of the service providers in an open-travel marketplace that may offer web-service functionality.

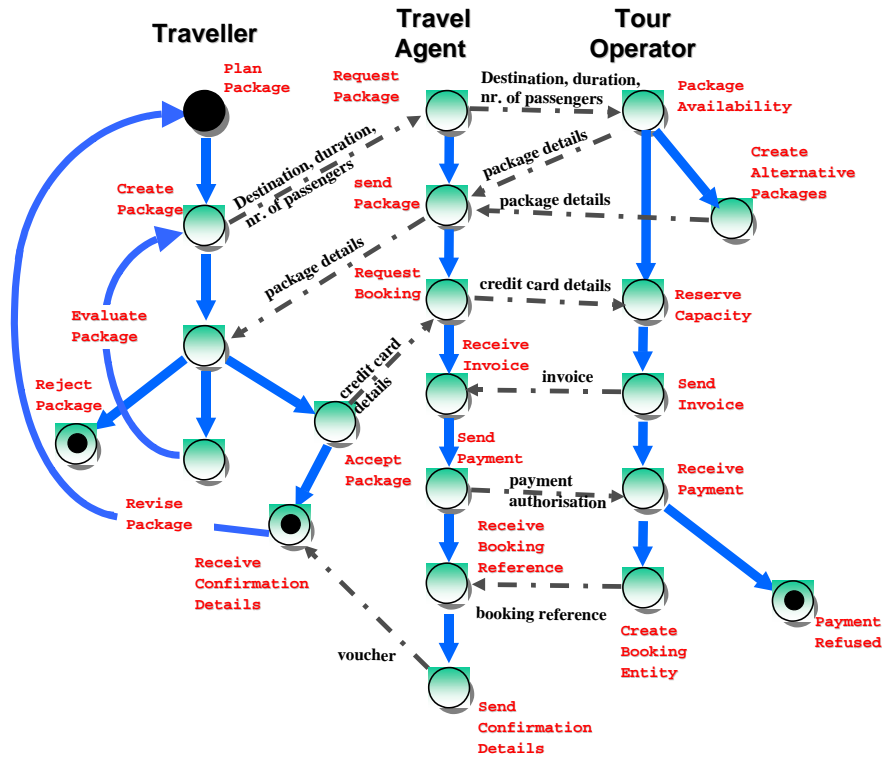


Fig. 1. The *Package Tours/Holiday Bookings* activity diagram.

We use an activity diagram to formally represent this standard business process, Figure 1. Solid arrows in this diagram represent flows of control while dashed arrows represent flow of messages. Each node in the diagram represents an activity. There are three agents whose activities are described: a traveller, a travel agent, and a tour operator. Following the UML convention we use solid filled circles to denote initial states and circles surrounding a small solid circle to denote end states. This formalism allows for cycles shown as outgoing arrows from end states.

In Figure 1, the business process is initiated by a traveller by providing the parameters necessary for a holiday package (destination, duration, nr. passengers). This triggers a request activity at some travel agent, which in turn triggers a package availability request from some tour operator. Holiday package related information messages are exchanged until the traveller is able to choose between rejecting, modifying or accepting the package. Potential users (travellers) can come up with requests to book their holidays on the basis of the business process described by this activity diagram. For instance, a traveller may desire to travel to Rome for a specific time frame and may also require a plane ticket and

optionally a hotel reservation for exactly the same period of time in the same city and for a given number of accompanying passengers.

Formalising the OTA specification with an activity diagram is just one of many possible options. Here we are not concerned with which is the best formalism for modelling web-service interactions, rather we are interested in presenting an architecture for interaction with e-marketplace based web-services that is independent from the chosen modelling formalism.

4 Service planning and constraint satisfaction

When users interact with e-marketplace based web-services, ideally they wish to specify only their goals and to obtain the desired results. Consequently, the basic premise of our approach is to equip the user with an expressive service request (goal) language so that he can specify his objectives clearly and unambiguously.

4.1 The request language

The services request language, described in this paper, is based on appropriate extensions of a formal planning language for expressing extended goals called *EaGLE* [6]. The *EaGLE* language is an extension of the well-known temporal logic CTL [8]. It inherits from CTL the possibility of declaring properties on the non-deterministic temporal evolutions of a system, and extends it with constructs that are useful for expressing complex planning goals, such as sequencing of goals, and goal preferences. *EaGLE* is defined over a set of activities A and constraint objects C as follows:

- $a(c)$ is a *basic well formed formula* (wff) if $a \in A$ and $c \in C$;
- $\neg a_1(c) \mid a_1(c) \wedge a_2(c') \mid a_1(c) \vee a_2(c')$ (c and c' are different constraint objects) and other logical combinations of basic wff are “basic” wff;
- **DoReach** $a_1(c) \mid$ **TryReach** $a_1(c) \mid$ **DoMaintain** $a_1(c) \mid$ **TryMaintain** $a_1(c)$ are wff iff $a_1(c)$ is a wff if $a_1(c)$ is a basic wff;
- g_1 **And** $g_2 \mid g_1$ **Then** $g_2 \mid g_1$ **Fail** $g_2 \mid$ **Repeat** g_1 are wff if g_1, g_2 are wff.

In the following we describe the most salient features of the *EaGLE* language (see [6] for a complete description and for the full semantics of the language).

EaGLE allows for expressing conditions that are *vital* to reach or to maintain (e.g., “**DoReach** $a(c)$ ”) or “**DoMaintain** $a(c)$ ”, respectively), as well as *optional* or *desired* conditions (“**TryReach** $a(c)$ ”) or “**TryMaintain** $a(c)$ ”).

Moreover, it allows for expressing *concatenation* of goals (e.g., “ $((a_1(c) \vee a_2(c'))$ **Then** $a_3(c'')$ ”) expresses the fact that activity $a_3(c'')$ has to happen after at least one of the activities $a_1(c)$ or $a_2(c')$ has occurred) and for expressing *preferences* between goals (e.g., “**TryReach** $a_1(c)$ **Fail** **DoReach** $a_2(c')$ ”) expresses the fact that the preferred goal is to achieve activity $a_1(c)$ but that, if this is not possible, then it is vital to achieve at least activity $a_2(c')$.

EaGLE is an expressively rich language to express goals over activities and succession of activities, but it lacks the possibility of expressing quantitative

information over individual activities, such as a timeout condition over a given activity. To overcome this limitation, we extend this language with the expressive power of linear constraints [9]. We associate with each activity a constraint object $c \in C$ that is composed of a number of variables that range over different domains and over which one can express linear constraints. More formally, if $c, c' \in C$ are constraint objects, $c.v_1, \dots, c.v_k$ a set of variables associated with the constraint object c , and D_1, \dots, D_n a set of domains equipped with operators, then we allow formulas of three kinds: first, variable declarations $c.v \in D_j$; second, linear constraint within a constraint object c involving variables, operators, and constants defined for D_j such that any two c variables comparing in the same constraint range over the same domain; third, linear constraint across constraint objects c and c' involving c and c' variables, operators and constants defined for D_j such that any two c and c' variables comparing in the same constraint range over the same domain.

The combination of the *EaGLE* with the linear constraint language forms a sound foundation for introducing constructs necessary for a web-services request language (WSRL). This language is used in the first instance for experimentation purposes and we plan to extend its functionality for completeness in the future.

4.2 The planning framework

A user can specify a request in terms of the WSRL introduced in the previous section by supplying the destination, duration, and number of passenger input parameters required to instantiate the formal business in Figure 2 and by attaching possible constraint to the request for a holiday package, e.g., flight prices, accommodation preferences, etc. An issue that arises is how to interpret the goal and how to plan an execution that satisfies it. This is the responsibility of the planning framework illustrated in Figure 2. This figure gives a high-level view of the planning framework. In particular, it illustrates that when a user formulates a request in the WSRL this is posed against a formal description of an abstract business model and business rules (top-left), e.g., the activity diagram of Figure 1. The parts of the formalised business process that are related to the user's request are then translated automatically into a state transition graph suitable for direct execution by a planner module. The planner checks whether the request can be satisfied. If the request can be satisfied, it returns a generic (uninstantiated) plan for further processing. If the request cannot be satisfied, then it furnishes an explanation of why the goal failed and prompts for reformulation of the request. Note that, at this level, the planner traverses activity paths along the formalised business process that could satisfy the request but it is not concerned with the values of the constraint objects tied to these activities, e.g., actual ticket prices, destinations and so on. We refer to this type of plan as a *generic plan*.

If there is a succession of activities that may satisfy the goal, then the generic plan is instantiated by interacting with UDDI registered web-services. This results in a number of *instantiated plans* for each generic plan. The instantiated

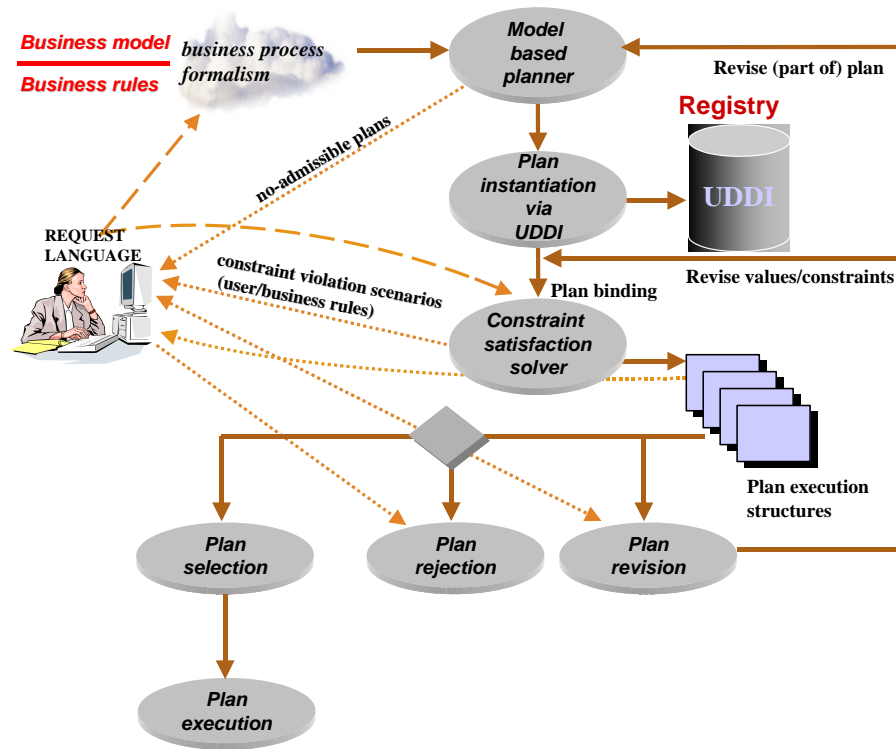


Fig. 2. Approach to planning for web-services.

plans provide values for the user's request such as different flight carrier possibilities, destinations, prices, duration, etc. For each instantiated plan a constraint satisfaction solver is invoked to check whether user specified constraints or web-service provider specified constraints are satisfied. If these constraints can be satisfied then the plan is considered as executable. If they are not, then the user is prompted again with information regarding the constraints that are violated for a particular instantiated plan. Finally, the constraint satisfaction module "approves" the set of *plan execution structures* that are ready for execution. At this point, the user has to make a choice. The user may select one plan from the set of plan execution structures, and commit to its execution. Alternatively, he may reject some or all the plans. The user may even choose to revise fragments of a set of executable plans. This results in a new planning phase at the level of the model based planner.

An experimental system for execution of service request and resulting plans is under construction. We have represented the formal model of the e-travel domain (activity diagram in Figure 1) in terms of NPDDL, a non-deterministic extension of the PDDL (the standard Planning Domain Description Language, see [10]).

The domain description and the goal (request), expressed in the *EaGLE* language enriched with constraints, was then fed to a Model Based Planning (MBP) software module to check the validity of the generic plan [11]. This is an advanced prototype system for execution of plans that either executes a plan in its entirety or returns an explanation of why the plan failed. Solving the linear constraints of the constraint objects coming both from the user side and from UDDI supplied web-service instances was achieved via *ECLiPSe*, a standard constraint satisfaction solver software module [12]. Communication between these two modules is at very preliminary stage, thus input data had to be simulated.

5 Example of executing a trip request

At this stage we revisit the OTA e-travelling example and consider a traveller that wants to travel to a destination in Italy such as Rome, Milan or Venice. This traveller wants to use an airline carrier, leave on the 1st of June and return on the 10th, with two accompanying passengers. These are the destination, duration, and number of passenger input parameters required by the formal business process in Figure 1. The traveller also requires accommodation for the same period of time and the same destination. Furthermore, the traveller finds it vital to have a flight ticket, but would still go even if s/he did not have a hotel reservation. Obviously the passenger does not wish to reserve accommodation without a confirmed flight ticket. In WSRL the request may look as follows:

(**DoReach** (Plan_Package(α) \wedge Receive_Confirmation(α)) **Then**
 (**TryReach** (Plan_Package(β) \wedge Receive_Confirmation(β))))

This request can be read as follows: "it is vital (**DoReach**) that I perform the activities in which I plan the package α (flight) and receive a confirmation for it. Once this vital activity is achieved (**Then**), optionally (**TryReach**), I want to plan the package β (hotel reservation) and receive a confirmation for it."

α and β represent a set of constraints over the e-travel domain described in the following:

$$\alpha. \begin{cases} \text{what} & \in \text{Services} \\ \text{arrival date} & \in \text{Dates} \\ \text{departure date} & \in \text{Dates} \\ \text{where} & \in \text{Cities} \\ \text{traveller} & \in \mathcal{N} \end{cases} \quad \beta. \begin{cases} \text{what} & \in \text{Services} \\ \text{arrival date} & \in \text{Dates} \\ \text{departure date} & \in \text{Dates} \\ \text{where} & \in \text{Cities} \end{cases}$$

$$\begin{array}{ll} \alpha.\text{what} & = \{\text{plane_ticket}\} & \beta.\text{what} & = \{\text{hotel}\} \\ \alpha.\text{arrival date} & = \{\text{May 1st}\} & \beta.\text{arrival date} & = \alpha.\text{arrival date} + 1 \\ \alpha.\text{departure date} & = \{\text{May 20th}\} & \beta.\text{departure date} & = \alpha.\text{departure date} - 1 \\ \alpha.\text{where} & \in \{\text{Rome, Milan}\} & \beta.\text{where} & \in \{\text{Rome, Venice}\} \\ \alpha.\text{traveller} & = 3 & \beta.\text{where} & = \alpha.\text{where} \end{array}$$

In the following we simulate the execution phases of this request in terms of the framework in Figure 2. We make the assumption that initial states generate

constraint objects that are then attached to all traversed activities or message links in the activity diagram. The objects are consumed by the end states.

Planning. The e-travelling example of Figure 1 is modelled in NPDDL by describing activities as actions of a non deterministic domain. The NPDDL description together with the goal of the user expressed in WSRL are to the MBP, which returns a plan.

Instantiation via UDDI. The plan returned by the model based planner MBP is instantiated by interacting with the UDDI registry. The same generic plan can be instantiated against various web-service providers.

Constraint satisfaction solver. Subsequently, the constraint objects with declarations and constrains supplied by both the user request and by interactions with the web-services, e.g., that the flight to Rome is not available on 1st June, need to be satisfied. The *ECLiPS* software module is used for this purpose. Each instantiated plan that satisfies the constraints is kept as candidate for execution. If there are no solutions to the constraint satisfaction problem, the user gets notified about which constraints are violated.

Plan choice. The output of the previous step is a series of plan execution structures which the user chooses.

6 Concluding remarks and future work

We have presented an architectural framework for web-service interaction based on planning and constraint satisfaction and a service request language developed on the basis of this framework. The design of the planning framework is based on coherent views of the issues arising when planning under uncertainty (as plans inevitably do not execute as expected) in dynamic environments where there is the constant need to be able to identify critical decision trade-offs, revise goals and evaluate alternative options. This framework recognises that in an uncertain and dynamic world such as that of web-services a correspondence must be drawn between the formal representation of this world and the planer's model of it. Consequently, the planning framework draws a correspondence between the formal representation of a business domain model and the planer's model of it. It then instantiates plans on the basis of the plan model via interaction with the UDDI infrastructure; provides a natural account of why plans are violated; and when necessary dynamically reconfigures plans on basis of user interaction. These design considerations are reflected at the level of the request language that generates plans over web-services residing in an e-marketplace.

We have given full semantics of the constructs of the WSRL that subsume the planning composition language we proposed in [7]. There we introduced constructs at the systems-level to support web-service composition, whereas the WSRL includes constructs such as alternative activities, vital vs. optional activities, preconditions, postconditions, invariants, and constraint operators over quantitative values that can be employed at the user-level when formulating a goal. The novelty of this approach lies in the automatic generation and verification of plans over web-services residing in an e-marketplace once a user request is

concretely expressed and formally specified. This framework provides genericity and flexibility and can be used as a basis for effective planning for different types of applications dealing with interacting web-services.

A preliminary implementation of the ideas mentioned in this paper has been constructed (as explained in section 4.2) and further tests are underway. A number of issues remain open for future investigation. First, activity diagrams are not expressive enough to model complex web-service interactions. For instance, they lack the ability of expressing cardinality constraints (such as that there are many travel agents for each traveller, or that there is exactly one payment for one e-voucher). Second, the integration between the *EaGLE* section and the constraint section of the WSRL is at an experimental stage and needs to be improved in order to be able to obtain more expressive plans. Furthermore, constraint formalisms especially tailored at electronic commerce can be used in place of linear constraints such as dynamic constraints [13].

References

1. OTA Open Travel Alliance. 2001C spec., 2001. <http://www.opentravel.org>.
2. S. Smith, D. Hildum, and D.R. Crimm. Toward the design of web-based planning and scheduling services. In *Int. Workshop on Automated Planning and Scheduling Technologies in New Methods of Electronic, Mobile and Collaborative Work*, 2001.
3. D. McDermott. Estimated-regression planning for interactions with Web Services. In *6th Int. Conf. on AI Planning and Scheduling*. AAAI Press, 2002.
4. C. A. Knoblock, S. Minton, J. L. Ambite, N. Ashish, I. Muslea, A. G. Philpot, and S. Tejada. The ariadne approach to web-based information integration. *International the Journal on Cooperative Information Systems*, 2002. Special Issue on Intelligent Information Agents: Theory and Applications, Forthcoming.
5. C. A. Knoblock, K. Lerman, S. Minton, and I. Muslea. Accurately and reliably extracting data from the web: A machine learning approach. *Data Engineering Bulletin*, 2002. To appear.
6. U. Dal Lago, M. Pistore, and P. Traverso. Planning with a language for extended goals. In *18th National Conference on Artificial Intelligence (AAAI-02)*, 2002.
7. J. Yang and M. Papazoglou. Web component: A substrate for web service reuse and composition. In *14th Int. Conf. on Advanced Information Systems Engineering CAiSE02*, 2002.
8. E. A. Emerson. Temporal and modal logic. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science, Volume B*. Elsevier, 1990.
9. P. Van Hentenryck and V.J. Saraswat, editors. *Principles and Practice of Constraint Programming*. MIT Press, 1995.
10. M. Ghallab, A. Howe, C. Knoblock, D. McDermott, A. Ram, M. Veloso, D. Weld, and D. Wilkins. PDDL—The Planning Domain Definition Language. In R. Simmons, M. Veloso, and S. Smith, editors, *Int. Conf. AIPS98*, 1998.
11. P. Bertoli, A. Cimatti, M. Pistore, M. Roveri, and P. Traverso. MBP: A Model Based Planner. In *n Proc. IJCAI'01 Workshop on Planning under Uncertainty and Incomplete Information*, 2001.
12. ECLIPSE. Eclipse Constraint Logic Programming System, 2002. <http://www-icparc.doc.ic.ac.uk/eclipse>.
13. M. Iwaihara. Supporting dynamic constraints for commerce negotiations. In *WECWIS 2000*. IEEE, 2000.