Reactive Web Agents with Open Constraint Programming

Kenny Q. Zhu Wee-Yeh Tan Andrew E. Santosa
Roland H.C. Yap
Department of Computer Science, National University of Singapore
Email: {kzhu,tanwy,andrews,ryap}@comp.nus.edu.sg

Abstract

This paper describes a new programming system for writing web applications with reactive agents, i.e. the agents can have complex responses which depend on how the environment changes. Our prototype system is based on the Open Constraint Programming framework using the Constraint Logic Programming language CLP($\mathcal{R}$). The benefit of reactive web agents is that activities of agents can be coordinated and synchronized using a common store and the agents can themselves be written as a system of interacting rules. Our thesis is that such a system makes it easy to write powerful reactive applications. We use a stock trading system to illustrate our reactive agents. Some details of the implementation are also given.

Keywords: web agents, multi-agent systems, constraint programming.

1 Introduction

Traditional web applications are client-server based (e.g. CGI, ASP, JSP), and typically involve relational database queries. With the increasing demand from e-commerce and online transactions, the next step in the evolution is that of reactive web based applications. By reactive, we mean that problem solving or query answering is triggered by events, i.e. it "reacts" to events. For example, in the auctioning of goods online, a user may launch an agent that requests for some merchandise at a particular price. If the product is not presently available, the user would usually prefer to have the agent remain with the auction system, auto-bidding if necessary, and notify the user when the bidding is completed. Without such a reactive agent, the user would have to repeatedly check and re-bid. There is also growing demands for sophisticated applications that require constraints, reasoning and knowledge representation. Problems like this with elements of reactivity and concurrency, can not be solved well with a traditional database transaction approach.

This paper presents a system for reactive constraint programming through web agents as an instance of the Open Constraint Programming (OCP) framework. We posit that such an OCP-based system has all the required elements which make it a suitable programming model for writing reactive web applications: (i) a global store which contains the current environment; (ii) rules to specify how applications should behave; and (iii) reactors which are actions which can be performed at some suitable future time which is provided by the synchronization mechanism in our web OCP system.

2 Related Work

Much research in inter-agent communications focuses on language aspects, such as in *Agents Communication Languages* (*ACLs*) [5], and the well-known contents language *Knowledge Interchange Format* (*KIF*) [1]. One difference is that in our work, *constraints* are part of the language of messages.

The closest relation to our work is *Business Rules for Electronic Commerce (BREC)* [2]. The overall objective of BREC is to develop a reusable technology for business rules and rule-based intelligent agents. Fundamental frameworks for business rules interoperability, their conflict handling, and procedural attachments are provided. Thus, BREC addresses the multi-agent problem from a different perspective, that is, the re-usability of rules and their consistency. Here instead, we focus on openness, programmability and reactivity.

3 Using OCP for Reactive Web Agents

Open Constraint Programming (OCP) is a framework which generalizes constraint stores and their interactions [3]. In particular, it addresses the notion of reactive agents which may have extended synchronization operations with a shared constraint store. The essential model consists of a shared constraint store, concurrent programmable agents which are independent of the store and a notion of a reactor which provides synchronization and atomicity of operations against the store. The most related work to OCP is Concurrent Constraint Programming [6] which has an elegant and clean semantics for programming concurrency. OCP differs from CCP in that we consider an open system of agents and the notion of a constraint store is much more general and need not be monotonic. By open, we mean that we do not restrict how the agents are programmed and new agents can interact with the store at any time. Reactive behavior occurs when some agents change the store and other agents need to react based on the change. Since OCP allows non-monotonic stores, it is necessary to have controlled mechanisms for atomic behavior of concurrently executing reactors.

In this paper, we describe the application of the OCP as a convenient system for writing web agents which have complex behaviors and need to react to changes in the shared constraint store. We target applications which require reasoning and knowledge representation. The constraint store here is a CLP program (this differs from CLP where the store are the constraints generated at runtime). This automatically gives us a database (the facts) and rules for reasoning about the state of the store.

The reactors are similar to CLP queries but with a synchronization condition and timeout. The prototype system uses the following simple reactors (more complex reactors are of course possible):

$$\begin{array}{l} \textbf{do } w_1|w_2|\cdots|w_n \Rightarrow action \\ \textbf{watching } t_1|t_2|\cdots|t_m \end{array}$$

Due to space constraints, we can only sketch the semantics of such reactors. The reactor consists of: synchronization conditions w_i which are just CLP goals which do not modify the environment themselves; an action(s) which is a CLP goal which may possibly modify the store; and watching conditions t_i are like w_i except that they may contain some query which involves the time. Intuitively, when the reactor is executed it waits until one of w_i is satisfied before executing $action$. However if one of the timeout conditions t_i is satisfied, the reactor exits. An example reactor from the stock trading application in Section 5 is:

```
do price(ibm, P), P < 80, cash(C),
  100*P < C ⇒ buy(ibm, P, 100)
watching expires(1 day)
```

This reactor waits for the price of IBM for 1 day to go under 80 and providing there is enough cash then buys it. The reactive behavior is dependent on the price of IBM, the amount of available cash and the timeout condition.

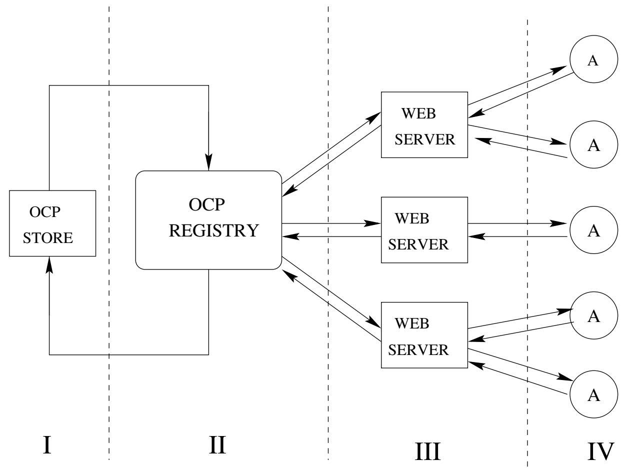

Figure 1. OCP's Four-Layered Architecture

4 The OCP System Architecture

The web agent based OCP system (OCPS) here has a four-layered architecture, shown in Figure 1.

I. Constraint store. The core of the structure is the *constraint store*, a shared memory of program agents, in the form of CLP programs. The relationship between the agents is represented by the constraints in the program. Here we use CLP($\mathcal{R}$) [4] as our solver and CLP system and thus reuse also the CLP($\mathcal{R}$) syntax.

II. Registry. As the CLP($\mathcal{R}$) system is sequential (as with most CLP systems), the second *registry* layer, remedies this limitation by providing concurrency control. The registry is responsible for the following tasks: (1) unpacks the requests from the reactors and translates them from OCP reactor language to a form that is understandable to the constraint solvers; (2) coordinates these requests and queues them up according to certain priority policy, and then feeds the requests to the solver; (3) maintains a *delayed queue* that contains all the requests that cannot be completed; (4) provides a trigger mechanism to facilitate wake-up of reactors according to an index table. Figure 2 shows a schematic of the OCP registry.

III. Web server. The third layer are the web servers. It is a medium between the OCP infrastructure and the user agents.

IV. User interface. The outer-most layer of our OCP web agent system is the user interface that translate user intention to the underlying reactor language. Here privileged users have the options to start or shutdown the OCP server at a host and a port, while normal users are able to submit their reactor agent in a user-friendly way.

With these four layers, we have the operational framework of the OCP reactive system. Concurrent reactive

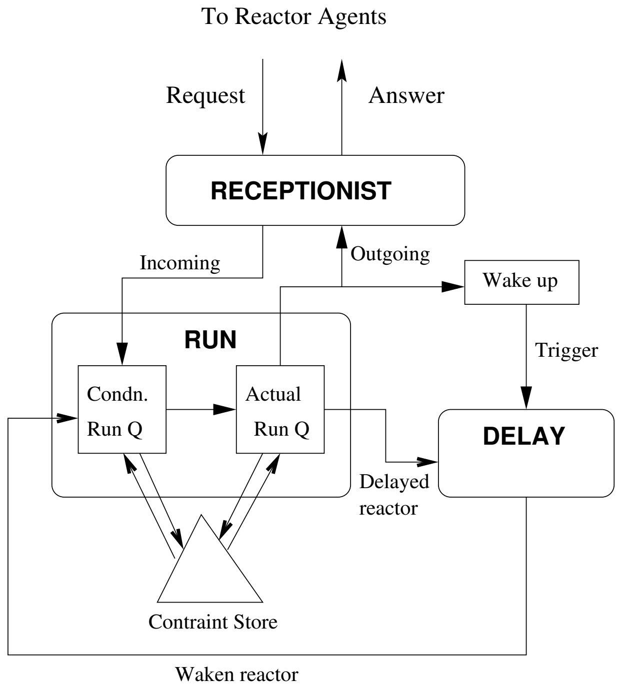

Figure 2. A Schematic of OCP Registry

agents can flow around in the system and make changes to the shared store if their blocking conditions are satisfied. Such changes impact the behaviors of other agents, especially those delayed ones. Agents are terminated if their requests have been answered or when the watch conditions become true.

5 Applications

5.1 Stock Trading System

To demonstrate our proposal, we have implemented a simplified stock trading system. This system consists of a number of clients that are front-ends used by human stock traders, an OCPS, and a stock market agent that keeps supplying the constraint store with changes in stock prices. OCPS has the information on stock prices (supplied by the external stock market agents) and all traders' portfolio. The human traders use the user interface of the clients to submit their trading strategies to OCP server.

The number of stocks owned is a basic fact that corresponds to a trader. If the trader "Gekko" owns the stocks with symbols ABC, DEF, GHI, JKL, MNO, and PQR, then the OCP server will store these facts in the context of the trader, for example:

```
stock(gekko,abc,30). stock(gekko,def,20).
stock(gekko,ghi,25). stock(gekko,jkl,30).
stock(gekko,mno,20). stock(gekko,pqr,250).
```

Using the above information and the stock prices, traders are free to submit their trading strategies to the OCP server in the form of mathematical constraints. Watchdogs are used to notify traders when some supplied facts conflict with their strategies, or to do actions (e.g., buying or selling of stocks) when the supplied facts enable them to so.

We rarely look at isolated items when talking about stocks, therefore, we provide a general mechanism for users to specify grouping of stocks, called *bundles*. The groupings help investors compare and contrast their holdings and diversify the investment.

For example, a trader may have defined two stock bundles:

- Low-risk stocks, including the stocks with ticker symbols ABC, DEF, GHI, JKL, and MNO.
- A high-risk stock with the ticker symbol PQR.

These are stored as the following facts in the server:

```
rating(gekko,abc,lowrisk).
rating(gekko,def,lowrisk).
rating(gekko,ghi,lowrisk).
rating(gekko,jkl,lowrisk).
rating(gekko,mno,lowrisk).
rating(gekko,pqr,highrisk).
```

In the system, there are also predefined bundles:

1. **System bundles**: These bundles consist of attributes that are provided by the system as defaults. They include:
   - **All**: All listed stocks.
   - **Cash**: The cash value of the trader, viewed as stocks.
   - Each stock, which is a bundle of its own.

   System bundles only change when new stocks are introduced or when stocks are removed.

2. **Administrator-predefined bundles**: These are volatile bundles that can be added as the market trend shifts. For example, if there are sufficient demand for dot-com stocks, the administrator may add a bundle that contains all the listed dot-com companies. These bundles are available only as a convenience to the users.

Having defined bundles, we may thus specify *strategies* that are to be applied on the bundles. Strategies are defined as mathematical constraints. A sample strategy is the one that is used to maintain a balance between high-risk and low-risk stocks. These are defined as the following constraints, with the function `port` computes the trader's portfolio of the stocks in a bundle.

```
100 <= port(lowrisk) (+/-10%),
port(highrisk) = 1*port(lowrisk) (+/-10%).
```

The first constraint states that the portfolio of low-risk stocks should not be less than $100. A ten percent tolerance is specified to state that the rule can be violated by ten percent. For example, a portfolio of $90 would be acceptable.

The second constraint says that the portfolio of high-risk stocks must balance the portfolio of low-risk stocks, again with ten percent tolerance.

At the time when a change in stock prices causes high-risk portfolio to be significantly greater than low-risk portfolio. A watchdog on the second constraint will notice this change, thus automatically perform the specified action, which could be the selling of high-risk stocks and/or the purchasing of low-risk stocks.

The determination of the stocks to purchase or sell can be guided by an *objective function*. For example the "minimize transaction cost" objective function will minimize the number of transactions to be performed. Other objectives such as user preferences can also be used.

5.2 Other Applications

Our web OCP system has other potential interesting applications, especially in the e-commerce where the transactions are volatile and may interact with one another. These applications can benefit from the constraint solving mechanism provided by OCP. They share the following properties:

- They consist of independent agents.
- The agents need to communicate in order to negotiate.
- There is need to reason from information provided by multiple agents.

Online auction. Auction is a process in which bidding price of an item keeps increasing at a potentially high frequency. Also the number of items on bid can be very large and one bidder is allowed to bid for or watch a number of offers at the same time. There is complex bid/offer relationship between the buyers and sellers. OCP offers a handy way to control concurrent bidding, and allows users to specify their bidding strategy, such as "if an MP3 player's price doesn't exceed $120 and a Company A's headphone cost less than $30, then I will bid for both and withdraw my bidding for the Company B's mini stereo."

Open market. In a generic marketplace, there are products, price, rules and there are buyers and sellers, and their finances. Purchasing from a market can be complicated if these elements are inter-related or constrained. For example, buying from supplier X will be cheaper in bulk but with more defects than supplier Y. This is a very typical optimization problem that a constraint based solution is most suitable for.

Timetable Schedulers. In this system a meeting chair specifies a meeting and invites all the participants to submit their preferences regarding the schedule of the meeting. The constraint solver in the back-end tries to solve the constraints submitted by the users. The OCPS thus inform the meeting chair and the meeting participants based on the information obtained.

6 Conclusion

We have introduced the structure of web agent OCP framework. We have also described how this framework supports the use of a CLP store in a multi-agent environment, both as the infrastructure and as a synchronization mechanism. The OCP framework in the context of web agents finds a number of useful applications in electronic commerce. Such problems which involve reactivity are better solved in our programming model than in traditional relational databases. Adapting traditional relational databases will require the use of active databases as well as a substantial programming effort to build in general reactivity. In contrast, the language of reactors being constraint logic programs, can be easily composed or modified to adapt to varying e-commerce scenarios. We have given an example using a stock portfolio management system where users can easily write their own reactors.

References

[1] Genesereth, M. R., "An Agent-Based Framework for Interoperability", In Bradshaw, J. M. *Software Agents*, Chapter 15, pp. 317–345.

[2] Grosof, B. N., Y. Labrou and H. Y. Chan. "A Declarative Approach to Business Rules in Contracts: Courteous Logic Programs in XML." In Wellman, M. P. ed. *Proceedings of the 1st ACM Conference on Electronic Commerce (EC '99), Denver, Colorado, USA, Nov. 3–5, 1999*. ACM Press, 1999.

[3] Jaffar, J. and R. Yap, "Open Constraint Programming, Invited Paper, *4th Intl. Conf on Principles and Practice of Constraint Programming (CP), Pisa*, Oct. 1998.

[4] Jaffar, J., S. Michaylov, P.J. Stuckey, and R.H.C. Yap, "The CLP($\mathcal{R}$) Language and System", *ACM Transactions on Programming Languages and Systems*, 14(3), 1992, pp. 339–395.

[5] Kone, M. T., A. Shimazu, and T. Nakajima. "The State of the Art in Agent Communication Languages". In *Knowledge and Information Systems 2(3)*, Aug. 2000.

[6] Saraswat, V. A., "Concurrent Constraint Programming". MIT Press, 1993.

[7] Wooldridge, M. and N. Jennings. "Intelligent agents: Theory and practice", *Knowledge Engineering Review*, 10(2), 1995.