

IMPLEMENTATION OF CONCURRENT CONSTRAINT TRANSACTION  
LOGIC AND ITS USER INTERFACE

A THESIS SUBMITTED TO  
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES  
OF  
MIDDLE EAST TECHNICAL UNIVERSITY

BY

FETHİ ALTUNYUVA

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS  
FOR  
THE DEGREE OF MASTER OF SCIENCE  
IN  
COMPUTER ENGINEERING

SEPTEMBER 2006

Approval of the Graduate School of Natural and Applied Sciences

\_\_\_\_\_  
Prof. Dr. Canan ÖZGEN  
Director

I certify that this thesis satisfies all the requirements as a thesis for the degree of Master of Science.

\_\_\_\_\_  
Prof. Dr. Ayşe KİPER  
Head of Department

This is to certify that we have read this thesis and that in our opinion it is fully adequate, in scope and quality, as a thesis for the degree of Master of Science.

\_\_\_\_\_  
Assist. Prof. Pınar ŞENKUL  
Supervisor

Examining Committee Members

Prof. Dr. Faruk POLAT	(METU)	_____
Assist. Prof. Dr. Pınar ŞENKUL	(METU)	_____
Assoc. Prof. Dr. Ali DOĞRU	(METU)	_____
Assoc. Prof. Dr. İsmail Hakkı TOROSLU	(METU)	_____
Assist. Prof. Dr. Murat ERTEN	(ETU)	_____

**I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.**

Name, Last name : Fethi ALTUNYUVA

Signature :

# **ABSTRACT**

## **IMPLEMENTATION OF CONCURRENT CONSTRAINT TRANSACTION LOGIC AND ITS USER INTERFACE**

Altunyuva, Fethi

MS., Department of Computer Engineering

Supervisor: Assist. Prof.Dr. Pınar Şenkul

September 2006, 49 pages

This thesis implements a logical formalism framework called Concurrent Constraint Transaction Logic (abbr.,CCTR) which was defined for modeling and scheduling of workflows under resource allocation and cost constraints and develops an extensible and flexible graphical user interface for the framework. CCTR extends Concurrent Transaction Logic and integrates with Constraint Logic Programming to find the correct scheduling of tasks that involves resource and cost constraints. The developed system, which integrates Prolog and Java Platforms, is designed to serve as the basic environment for enterprise applications that involves CCTR based workflows and schedulers. Full implementation described in this thesis clearly illustrated that CCTR can be used as a workflow scheduler that involves not only temporal and causal constraints but also resource and cost constraints.

Keywords: Concurrent Transaction Logic, Workflow Scheduling, Constraint Logic Programming, Resource Allocation

# ÖZ

## KOŞUT ZAMANLI KISIT HAREKET MANTIĞI ÇİZELGESİ GERÇEKLEŞTİRMESİ VE KULLANICI ARA YÜZÜ GELİŞTİRİLMESİ

Altunyuva, Fethi

Yüksek Lisans, Bilgisayar Mühendisliği Bölümü

Tez Yöneticisi: Yrd.Doç.Dr. Pınar ŞENKUL

Eylül 2006, 49 sayfa

Bu tez maliyet ve kaynak tahsisine bağlı iş akışlarının modellemesi ve çizelgelenmesi için tanımlanan ve koşut zamanlı kısıt hareket mantığı çizelgesi (KZKHMÇ) adı verilen mantıksal formülleme çatısını gerçekleştirmekte, bu çatı için genişletilebilir ve uygun bir kullanıcı ara yüzü geliştirmektedir. KZKHMÇ, maliyet ve kaynak kısıtları içeren işlemler için bir çizelgeleme bulma amacıyla koşut hareket mantığını genişletmiş ve Kısıt Mantık Programlamasına entegre olmuştur. Prolog ve Java platformları entegre edilerek geliştirilen sistem, KZKHMÇ tabanlı iş akışları ve çizelgelemeler ihtiva eden iş uygulamaları için temel ortam hizmeti sunacak şekilde tasarlanmıştır. Bu tezdeki gerçekleştirmeler KZKHMÇ'nin sadece geçici ve sebep kısıtları için değil aynı zamanda maliyet ve kaynak tahsisi kısıtları gerektiren iş akışların çizelgelemesi için de kullanılabileceğini göstermiştir.

Anahtar Sözcükler: Koşut Zamanlı Hareket Mantığı, İş Akışı Çizelgeleme,Kısıt Mantık Programlama, Kaynak Tahsisi.

## **ACKNOWLEDGMENTS**

I would like to present my special thanks to my supervisor Assist.Prof. Dr. Pinar Şenkul for her supervision, guidance, and understanding throughout the development of this thesis.

I would like to thank Assoc. Prof. Dr. İ.Hakkı Toroslu for his suggestions and comments.

I would also thank to my wife, Hilal, for his great support and encouragement.

# TABLE OF CONTENTS

<b>ABSTRACT .....</b>	<b>IV</b>
<b>ÖZ .....</b>	<b>VI</b>
<b>ACKNOWLEDGMENTS .....</b>	<b>VIII</b>
<b>TABLE OF CONTENTS.....</b>	<b>IX</b>
<b>LIST OF TABLES .....</b>	<b>XII</b>
<b>LIST OF FIGURES .....</b>	<b>XIII</b>
<b>LIST OF ABBREVIATIONS.....</b>	<b>XIV</b>
<b>CHAPTER</b>	
<b>1.INTRODUCTION.....</b>	<b>1</b>
1.1. MOTIVATION AND SCOPE .....	2
1.2. ARCHITECTURE OF THE PROPOSED CCTR ENVIRONMENT .....	3
1.3. ORGANIZATION OF THE THESIS .....	3
<b>2.CONCURRENT CONSTRAINT TRANSACTION LOGIC.....</b>	<b>5</b>
2.1. OVERVIEW OF TRANSACTION LOGIC PROGRAMMING .....	6
2.2. OVERVIEW OF CONCURRENT TRANSACTION LOGIC PROGRAMMING .....	6
2.3. OVERVIEW OF CONCURRENT CONSTRAINT TRANSACTION LOGIC PROGRAMMING .....	7

<b>3.THE CCTR ENVIRONMENT .....</b>	<b>13</b>
3.1. DESIGN GOALS .....	14
3.2. EVALUATION OF UTILIZED TECHNOLOGIES .....	15
<b>4.USER INTERACTION AND GRAPHIC DESIGN.....</b>	<b>17</b>
4.1. CCTR ENVIRONMENT INTRODUCTION .....	17
4.2. STARTING CCTR ENVIRONMENT .....	19
4.3. CCTR APPLICATION .....	20
4.4. MENU OVERVIEW .....	21
4.5. CREATING CCTRFLOW DIAGRAMS.....	22
4.6. RUNNING CCTRFLOWS .....	26
4.7. SHOWING RESULTS.....	26
<b>5.DESIGN AND IMPLEMENTATION.....</b>	<b>28</b>
5.1. APPLICATION CONTROL AND GUI(JAVA PART).....	28
5.2. CCTR SCHEDULER(PROLOG PART) .....	30
<b>6.CONCLUSIONS .....</b>	<b>36</b>
<b>REFERENCES .....</b>	<b>37</b>

## **APPENDICES**

<b>A.TUTORIAL:STARTING CCTR ENVIRONMENT .....</b>	<b>39</b>
<b>B.TRANSFORMER GRAMMER.....</b>	<b>42</b>
<b>C.SAMPLE WORKFLOWS.....</b>	<b>42</b>

## LIST OF TABLES

<b>TABLE 2.1:</b> TRANSFORMATION RULES. ....	11
<b>TABLE 4.1:</b> EXAMPLE 4.1 CONSTRAINTS TABLE.....	19
<b>TABLE 4.2:</b> CCTRFLOW PALETTE ELEMENTS. ....	23
<b>TABLE 5.1:</b> CCTRFLOW PALETTE ELEMENTS. ....	29
<b>TABLE 5.2:</b> MODIFIED TRANSFORMATION RULES.....	32
<b>TABLE 5.3:</b> TRANSFORMER PARTIAL PREDICATES EQUIVALENCES .	34
<b>TABLE 8.1:</b> SAMPLE 1 CONSTRAINTS TABLE.....	47
<b>TABLE 8.2:</b> SAMPLE 1 SOLUTION TABLE. ....	48
<b>TABLE 8.3:</b> SAMPLE 2 CONSTRAINTS TABLE.....	49
<b>TABLE 8.4:</b> SAMPLE 2 SOLUTION TABLE. ....	49

## LIST OF FIGURES

<b>FIGURE 1.1:</b> HOUSE CONSTRUCTION CCTRFLOW.....	3
<b>FIGURE 2.1:</b> OVERVIEW OF TR,CTR,CCTR. ....	5
<b>FIGURE 2.2:</b> THE BIG PICTURE.....	10
<b>FIGURE 3.1:</b> ARCHITECTURAL OVERVIEW OF CCTR ENVIRONMENT. ....	13
<b>FIGURE 4.1:</b> CONFIGURATION WINDOW.....	20
<b>FIGURE 4.2:</b> MAIN WINDOW OF CCTR ENVIRONMENT.....	21
<b>FIGURE 4.3:</b> MAIN MENU OF CCTR ENVIRONMENT.....	21
<b>FIGURE 4.4:</b> DEFINE RESOURCES WINDOW .....	25
<b>FIGURE 4.5:</b> DEFINE RESOURCE COSTS WINDOW .....	25
<b>FIGURE 4.6:</b> HOUSE CONSTRUCT CCTRFLOW DIAGRAM.....	26
<b>FIGURE 4.7:</b> HOUSE CONSTRUCT CCTRFLOW RESULTS.....	27

## **LIST OF ABBREVIATIONS**

WfMS	:	Workflow Management System
TR	:	Transaction Logic
CTR	:	Concurrent Transaction Logic
CCTR	:	Concurrent Constraint Transaction Logic

# CHAPTER 1



## INTRODUCTION

A workflow is defined as coordinated set of activities that act together to achieve a well-defined goal. Trip planning, catalog ordering and manufacturing process of enterprises are typical examples of workflows [7]. While achieving a goal workflow is subject to various constraints that reflect the business logic of an enterprise. In enterprise applications it is needed to find a correct execution sequence of workflows that obeys the constraints logic of a workflow. Finding a correct sequence of execution is called *workflow scheduling*. There has been much research on workflow scheduling [9, 10, 11, 12]. However most of these researches are mainly focused on temporal and causality constraints, which aim to find correct ordering of tasks. A temporal/causality constraint is typically of form “*task 1 precedes task 2*” or “*if task 1 executes then tasks 2 and 3 must execute as well*”. In these forms of constraints resource allocations constraints and cost constraints are ignored [7]. In contrast, business applications frequently require resource allocations constraints and cost constraints. For example a company’s personnel or physical objects may not be allocated to the same task simultaneously or there might be a limit on the cost of using these resources. To fill this gap, a logical framework called Concurrent Constraint Transaction Logic which extends Concurrent Transaction Logic is proposed in [8]. Proposed logical formalism specifies, verifies, and schedules workflows subject to resource and cost constraints [8].

## 1.1.Motivation and Scope

This thesis implements the proposed logical formalism framework in [8] and develops an extensible and flexible graphical user interface for the proposed framework. The developed system, called CCTR Environment which integrates Prolog and Java Platforms, is designed to serve as the basic environment for modeling enterprise applications that involves CCTR based workflows and schedulers. Developed environment containing CCTR Scheduler and its user interface is designed in such a way that can be an alternative for workflows schedulers that requires not only temporal and causal constraints but also resource and cost constraints.

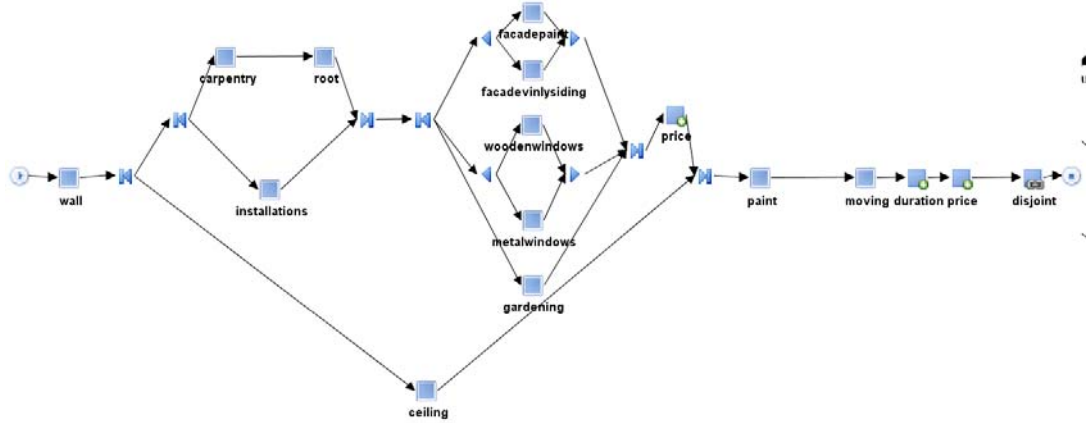
The motivating example defined below will be followed in this thesis. The example derived from [13] and used in [8].

**Example 1.1** “Company A builds a house, does gardening, and moves customer’s furniture into the new house. The company subcontracts various tasks to other companies. There are several candidate subcontractors for each task and some subcontractors may qualify to perform several subtasks. Company A wants to choose subcontractors in a way that satisfies the customer’s requirements and so that the costs (in time and money) do not exceed a previously agreed upon estimate. The CCTRflow diagram is shown in Figure 1.1 which designed in CCTRflow Designer. In the figure, the  represents branches of work that can be done in parallel (and all must be executed).  represents alternative courses of action where only one of the branches needs to be executed. For instance, the facade can be painted or the customer might choose to use vinyl siding (but not both). Tasks that must be done in sequence are connected via directed edges. Resource related constraints for this workflow can include the following:

- The budget for the entire project should not exceed \$30,000.
- The project should not last longer than 10 days.
- Different subcontractors must be chosen for tasks that are determined to be parallel.

- In addition, it is required that the cost of building the facade, installing windows, and doing gardening should not exceed \$15,000.”[8]

The goal is to develop an environment for defining above CCTRflow and to find valid schedule(s) for it. The schedule should include an execution order of the entire subtask plus an assignment of the resources to tasks so that all the constraints are satisfied.



**Figure 1.1:** House Construction CCTRflow.

## 1.2. Architecture of the Proposed CCTR Environment

The proposed architecture of CCTR Environment is constituted by integrating Java Platform[14], XSB Platform[15] and SWI Platform[16]. Java Platform controls the application logic and provides Graphical User Interface. Implementation of CCTR Scheduler is developed on XSB Platform. Finally, SWI Platform is used for the implementation of Constraint Solver. Details of architecture are presented in chapter 3 of this thesis.

## 1.3. Organization of the thesis

The thesis is organized as follows:

**Chapter 2** contains an overview for Transaction Logic and Concurrent Transaction Logic and describes Concurrent Constraint Transaction Logic.

**Chapter 3** describes developed CCTR Environment, and design goals, and explains the utilized technologies.

**Chapter 4** presents the graphical design developed for the CCTR Environment

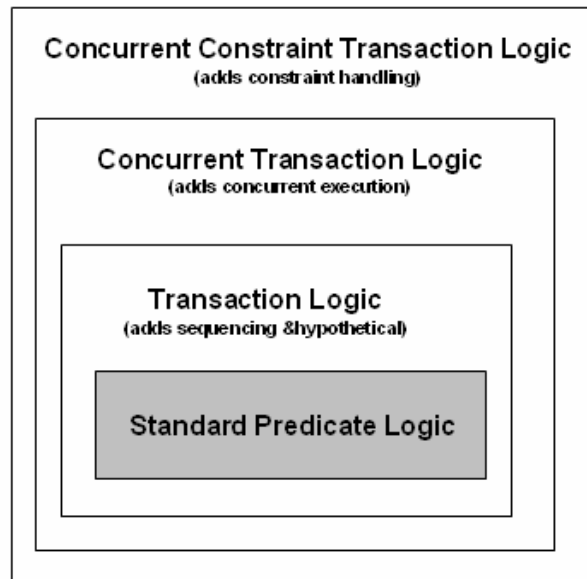
**Chapter 5** explains design and implementation details.

**Chapter 6** summarizes and concludes the thesis.

## CHAPTER 2

# CONCURRENT CONSTRAINT TRANSACTION LOGIC

In this chapter, a brief overview of Transaction Logic Programming (abbr,TR), Concurrent Transaction Logic (CTR) and Concurrent Constraint Transaction Logic(abbr,CCTR) is given. Relation among TR, CTR, and CCTR is depicted in Figure 2.1.



**Figure 2.1:** Overview of TR,CTR,CCTR.

## 2.1. Overview of Transaction Logic Programming

Transaction logic is extension of predicate logic proposed by Bonner and Kifer[1], which accounts for the state change of logic programs and databases in a declarative fashion. Model theory and complete proof theory of Transaction Logic is presented in [1]. Main contribution of Transaction Logic to predicate logic is its ability to program transactions. It is possible to program transactions in Transaction Logic because Horn version of TR has both procedural and declarative semantics.

The most important operator that TR extends the syntax of first-order logic is the binary operator,  $\otimes$ , called *serial conjunction*. The formula  $\psi \otimes \phi$  means “First execute transaction  $\psi$  then execute transaction  $\phi$ .” To build a wide variety of formulas, TR combines serial conjunction operator with the standard logical operators such as  $\neg$ ,  $\wedge$ , and  $\vee$ . By combining simple actions into complex ones TR increases its expressiveness and supports wide range of functionality in different application areas such as database queries and updates, bulk updates, transaction definition, deterministic and non-deterministic transactions, conditions on actions etc. [2].

Hung in [2] implemented Transaction Logic and evaluated the performance in different environments. Hung developed four prototypes ranging from slow but simple ones to fast but complex ones. The results of Hung’s work show that in some implementations, Transaction Logic programs achieve more efficiency comparable to programs with destructive updates in procedural programming languages [2].

## 2.2. Overview of Concurrent Transaction Logic Programming

CTR is an extension of TR which models concurrent execution of complex processes by introducing new connectives. Concurrency feature of CTR increases the flexibility, performance and power of the language [3]. CTR integrates concurrency, communication, and database updates in a completely logical framework. [3] This feature of CTR makes it unique among the other deductive databases. Natural model theory and proof theory of CTR can be found in [4, 5].

CTR, covers concurrent access to shared resources, communication between sequential processes, and isolating the inner workings of a group of processes from the outside world which many of them are features of process algebras [6]. For example in [4] it is explained that CTR is compositional that is processes are defined recursively in terms of sub-processes. This means that it is possible to specify multi-level processes, even when the number of levels is determined at runtime. In [4] it is also explained that, CTR provides high-level support for database functions. These include declarative queries bulk updates, views, and serializability. CTR also has many features of advanced transaction models, including sub-transaction hierarchies, relaxed ACID requirements and fine-grained control over abort and rollback [19]. This integration of process modeling and database functionality is reflected in the formal semantics of CTR, which is based on both database states and events, while the semantics of process algebras is based entirely on events [3]. Most recent implementation of CTR including two optimizations are presented in [3].

### **2.3. Overview of Concurrent Constraint Transaction Logic Programming**

Concurrent Constraint Transaction Logic is extension of Concurrent Transaction Logic (CTR) which integrates CTR with Constraint Logic Programming. CCTR can be used to model and schedules workflows that obey wide range of resource allocation problems [8].

#### **2.3.1 Syntax**

“The alphabet of CCTR consists of four countable sets of symbols: a set  $F$  of function symbols, a set  $V$  of variables, a set  $P$  of (regular) predicate symbols and a set  $C$  of *constraint predicates*. Each function, predicate and constraint predicate symbol has an *arity*, which indicates the number of arguments the symbol takes. Constants are viewed as 0-arity function symbols and propositions are viewed as 0-arity predicate symbols. A term is a constant or has the form  $f(t1, ..., tn)$ , where  $f$  is a function symbol of *arity*  $n$ , and  $t1, ..., tn$  are terms.”[8]

CCTR models workflows and constraints using the rule-based paradigm, and workflows are represented using only a subset of CCTR: rules and goals.

**Definition 2.3.1** *CCTR goals* are recursively defined as follows:

- *Atomic goal*:  $p(t_1, \dots, t_n)$ , where  $p \in P$
- *Serial goal*:  $\phi_1 \otimes \dots \otimes \phi_2$ , where each  $\phi_i$  is a CCTR goal
- *Parallel goal*:  $\phi_1 \mid \dots \mid \phi_2$ , where each  $\phi_i$  is a CCTR goal
- *Disjunctive goal*:  $\phi_1 \vee \dots \vee \phi_2$ , where each  $\phi_i$  is a CCTR goal
- *Constraint goal*:  $\phi_1 \wedge \text{constr}$ , where each  $\phi_i$  is a CCTR goal and *constr* is a Boolean expression.

We can define House Construction example in section 1.2 in CCTR Syntax as follows.

$\text{wall} \otimes (((\text{carpentry} \otimes \text{roof}) \mid \text{installations}) \otimes \text{the-middlepiece}) \mid \text{ceiling})$   
 $\otimes \text{paint} \otimes \text{move}$

The middle piece defined separately as :

$(\text{façade-paint} \vee \text{façade-vinyl}) \mid (\text{wooden-windows} \vee \text{metal-windows}) \mid$   
 $\text{gardening}$

### 2.3.2 Semantics.

In CTR, serial and concurrent execution is modeled as m-paths which is not sufficient to model resource requirements of CCTR. For this purpose *partial schedule* is introduced in [8] which adds more structure to m-paths. Partial schedules are defined in terms of two operators:  $\bullet_p, \parallel_p$

**Definition 2.3.2** A *partial Schedule* is defined as follows:

- An *m-path*,  $\pi$ , is a *partial schedule*
- *Serial composition* of two partial schedules,  $\omega_1 \bullet_p \omega_2$ , is a *partial schedule*
- *Parallel composition* of two partial schedules,  $\omega_1 \parallel_p \omega_2$ , is a *partial schedule*

Other notations introduced by CCTR for defining and solving resource allocation constraints are *resource* and *resource assignments*.

**Definition 2.3.3** *A resource is an object that represents a physical or abstract resource that is needed for a workflow to execute. It is assumed that each resource has one or more associated use costs of different kinds, and that these costs are represented by the attributes of the resource.*

**Definition 2.3.4** *A resource assignments is a partial mapping from partial schedules to sets of resources. Any resource assignment, must satisfy the following conditions:*

- $\mathcal{A}(\omega_1 \bullet_p \omega_2) = \mathcal{A}(\omega_1) \cup \mathcal{A}(\omega_2)$ , if both  $\mathcal{A}(\omega_1)$  and  $\mathcal{A}(\omega_2)$  are defined
- $\mathcal{A}(\omega_1 \parallel_p \omega_2) = \mathcal{A}(\omega_1) \cup \mathcal{A}(\omega_2)$ , if both  $\mathcal{A}(\omega_1)$  and  $\mathcal{A}(\omega_2)$  are defined

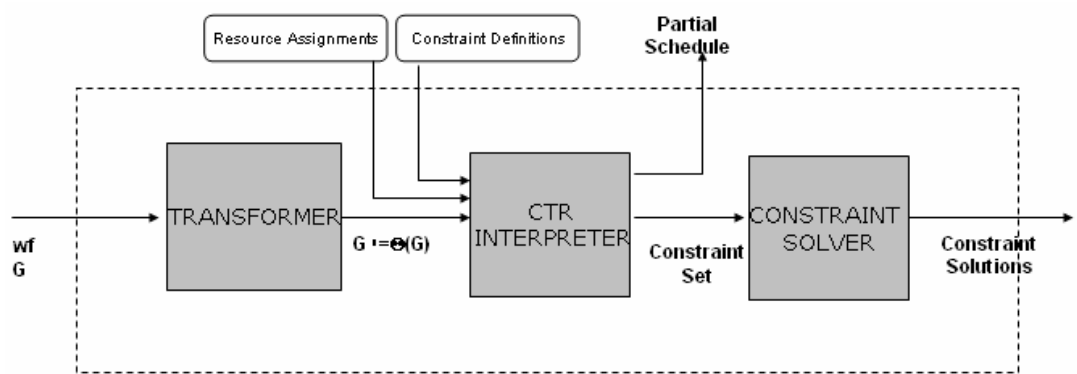
Last definition needed to complete the CCTR semantics is constraint universe definition.

**Definition 2.3.5** *A constraint universe  $\mathcal{D}$  is a set of domains together with some relations over these domains.*

The domains in a constraint universe include scalar domains such as integer, float, or string and special CCTR domains of partial schedules, resources, resource assignments.

### 2.3.3 CCTR as Workflow Scheduler.

The algorithm for scheduling workflows subject to cost and resource algorithms is as follows: A CCTR goal is formed that specifies the control flow of a workflow plus resource allocation and cost constraints. CCTR goal is transformed into a CTR goal where resource allocation and costs are incorporated as postconditions rather than conjuncts. Since output goal does not contain  $\vee$  logical connective it can be executed using CTR[3] proof theory. Execution of CTR finds partial schedules and generates constraint set for constraint solver. Constraint solver determines a complete schedule of tasks that is consistent with the given goal and outputs the resource assignment that obeys the resource and cost constraints. Architecture of CCTR Scheduler is shown in Figure 2.2.



**Figure 2.2:** The Big Picture.

### 2.3.4 Transformation Rules

As stated before purpose of the transformation step is to get a CTR formula so that it can be evaluated by using the proof theory of [3]. The transformation rules are shown in Table 2.1. For the simplicity of notation it is assumed that cost and resource constraints are represented by a single atomic formula of the form  $c(t_1 \dots t_2)$ , where  $c \in C$ .

**Table 2.1:** Transformation Rules.

	$\Theta(G): \rightarrow \Theta_1(\text{DNF}(G))$ ; $\Theta_1$ is described below.
	<b>Transformation <math>\Theta_1</math></b> ; uses auxiliary transformation $\Theta_2$
(1)	$\Theta_1(A): \rightarrow A \otimes \text{resource}(A, \text{Resources}_A)$ , if $A$ is a an atomic formula <i>resource</i> is the assignment predicate; $\text{Resources}_A$ is a new variable
(2)	$\Theta_1(G \wedge \text{constr}): \rightarrow \Theta_1(G) \otimes \text{constr}(\Theta_2(G))$ , $\text{constr} \in C$ is constraint predicate and <u>constr</u> is a new predicate that simulates $C_D$ - interpretation of $c$ in constraint universe $D$ . $\Theta_1$ is defined below
(3)	$\Theta_1(G_1 \text{ op } G_2): \rightarrow \Theta_1(G_1) \text{ op } \Theta_1(G_2)$ , if $\text{op}$ is $ $ , $\otimes$ , or $\vee$
	<b>Auxiliary transformation <math>\Theta_2</math></b> ;
(1)	$\Theta_2(A): \rightarrow \text{Resources}_A$ , if $A$ is an atomic formula $\text{Resources}_A$ is the same variable that was used in step (1) of $\Theta_1$ .
(2)	$\Theta_2(G \wedge \text{constr}): \rightarrow \Theta_2(G)$ , where $\text{constr} \in C$ is a constraint predicate
(3)	$\Theta_2(G_1 \otimes G_2): \rightarrow \bullet_p(\Theta_2(G_1), \Theta_2(G_2))$
(4)	$\Theta_2(G_1   G_2): \rightarrow \parallel_p(\Theta_2(G_1), \Theta_2(G_2))$

### 2.3.5 Constraint Definitions

Definition of new predicate constr introduced in transformation rules depends on the semantics of constr, so it is not possible to define a general transformation that will adjust all cases. However using common properties of resource and cost constraints we can define templates for them.

#### Resource Template:

$\text{rsrc\_constr}(\text{Args}, \bullet_p(G_1, G_2)) \rightarrow \text{rsrc\_constr}(\text{Args}, G_1), \text{rsrc\_constr}(\text{Args}, G_2),$

$\text{test\_rsrc}_{\text{serial}}(\text{Args}, G_1, G_2)$

$\text{rsrc\_constr}(\text{Args}, \parallel_p(G_1, G_2)) \rightarrow \text{rsrc\_constr}(\text{Args}, G_1), \text{rsrc\_constr}(\text{Args}, G_2),$

$\text{test\_rsrc}_{\text{parallel}}(\text{Args}, G_1, G_2)$

**Cost Template:**

$\text{cost\_constr}(\text{Args}, G) \rightarrow \text{aggregate\_cost}(\text{Args}, \text{Cost}), \text{cost\_test}(\text{Args}, \text{Cost})$

$\text{aggregate\_cost}(\text{Cost}, \bullet_p(G_1, G_2)) \rightarrow$

$\text{aggregate\_cost}(\text{Cost}, G_1), \text{aggregate\_cost}(\text{Cost}, G_2),$

**combine<sub>serial</sub>**  $(\text{Cost}, G_1, G_2)$

$\text{aggregate\_cost}(\text{Cost}, \parallel_p(G_1, G_2)) \rightarrow$

$\text{aggregate\_cost}(\text{Cost}, G_1), \text{aggregate\_cost}(\text{Cost}, G_2),$

**combine<sub>parallel</sub>**  $(\text{Cost}, G_1, G_2)$

**2.3.6 CTR Interpreter**

The transformed workflow is passed to the CTR interpreter [3], which tries to find a valid execution of the workflow using the inference rules of the CTR proof theory. Any path produced by the CTR Interpreter is a valid serialization of the workflow execution. Constraint solving process of CCTR Scheduler needs to obtain partial scheduler structure from CTR interpreter. In order to capture partial schedule structure from CTR Interpreter, goal transformation process creates additional argument, and CTR Interpreter defines a new predicate, called *schedule(Task)*. Details are given in Chapter 5 of this thesis.

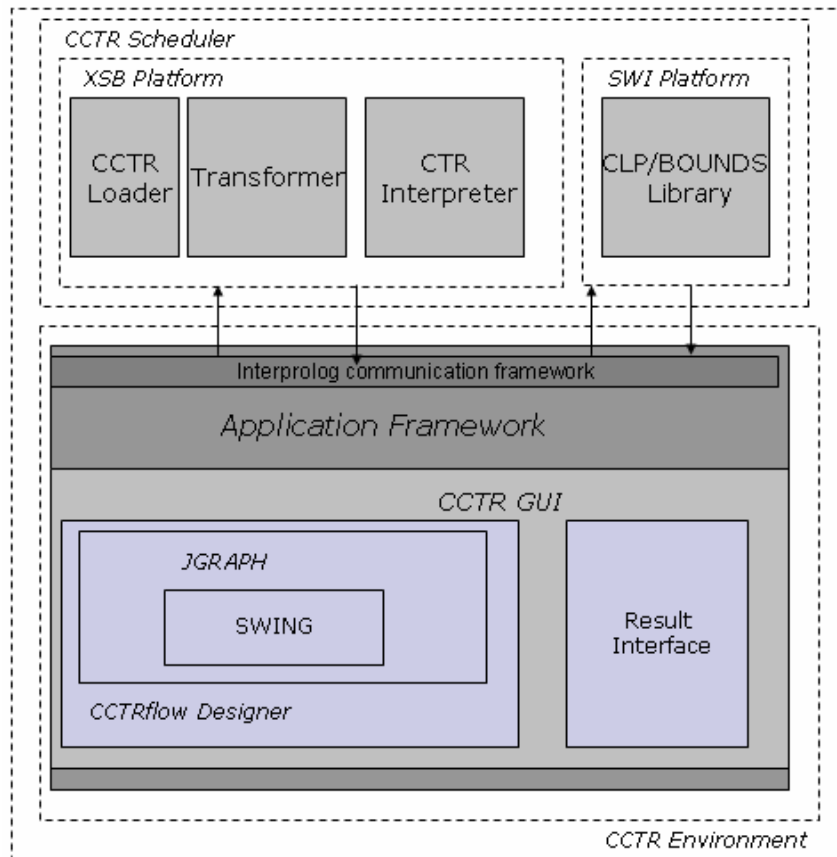
**2.3.7 Constraint Solver**

The role of Constraint Solver is to process the constraints that are found by CTR Interpreter while it searches for valid execution of the given workflow. Any solution to the constraint set is a valid resource assignment for the workflow schedule produced by the interpreter [8]. In the implementation constraint solver library of SWI is used. Although CTR Interpreter is a XSB application, SWI is used to show that any off-the-shelf constraints solver that is compatible with the defined constraints can be used.

## CHAPTER 3

### THE CCTR ENVIRONMENT

CCTR Environment is constituted by integrating three platforms: *Java platform*, *XSB Platform*, and *SWI Platform*. Figure 5 depicts the general architecture and gives an overview of the interaction between these platforms:



**Figure 3.1:** Architectural overview of CCTR Environment.

**Java Platform:** Application framework of *CCTR Environment* is developed on the Java Platform. All the user interactions are controlled by the GUI developed on

the top of this platform. Main components of CCTR GUI are CCTRflow Designer and Result Interface. CCTRflow Designer is a “What You See What You Get” (WSYWYG) designer specially developed for CCTR Environment. It enables to create CCTRflows with drag and drop facilities in an easy way to the user. The execution of CCTRflow is tracked by *Result Interface*. Interactions with other platforms are also managed from the java platform with the help of Interprolog communication framework.

**XSB Platform:** XSB Platform is the platform that holds the CCTR Interpreter. XSB platform executes any CCTRflow with constraints assignments provided by Java Platform and finds partial schedules and creates its corresponding constraint sets for SWI platform.

**SWI Platform:** SWI platform tries to find a solution for constraint set provided by XSB platform via Java Platform and return the results to Java Platform to represent to the user.

The detail of the implementation is explained Chapter 5 of this thesis.

### 3.1.Design Goals

The main goal of this thesis is to implement CCTR Interpreter and develop an extensible and flexible graphical user interface for the CCTRflows. On the other hand achieving this goal requires to integrate different systems in different platforms. In this respect, developed CCTR Environment must be integrated in a way that gives users the feeling of using a single application. Beside the demands of the users, it must also meet the demands of application programmers who wish to extend the system and of third parties who wish to customize the system.

Consequently main design goals in this thesis can be summarized as:

**Conformance to standards:** The GUI must conform to the Java Look and Feel Guidelines. It should also be similar to other applications the users know from similar domains.

**Models and mappings:** The GUI must provide a good conceptual model of the CCTR Framework. It must support the user by providing natural mappings between theory and implementation.

**Accessibility:** All important functions should be accessible easily. It should give user to configure some of the options.

**Extensibility:** The Application framework must provide a simple way to extend the functionality of CCTR.

**Platform Independency:** The developed CCTR environment must be available in a platform independent way. It should follow the promise “write once, run anywhere”

### 3.2.Evaluation of Utilized Technologies

In this section, some of the technologies that are selected for the implementation of CCTR Environment, according to design goals above will be explained. Main technologies used in the implementation are JAVA J2SE[14], JGraph[17], InterProlog[18], XSB[15] and SWI[16].

**JAVA J2SE:** “Java Platform, Standard Edition (also known as Java 2 Platform) lets to develop and deploy Java applications on desktops and servers, as well as today's demanding Embedded and Real-time environments”[14]. Java is the name of programming language used by Java Platform. “Java is a simple, object-oriented, network-savvy, interpreted, robust, secure, architecture neutral, portable, high-performance, multithreaded, dynamic language” [14]. The ability to write code that can be executed on any platform which has a Java virtual machine ported to it is the most significant reason for the selection of Java Platform as the implementation platform of CCTR Environment.

**JGraph:** “JGraph is nature, feature-rich open source graph visualization library written in Java. JGraph is a swing compatible library, both visually and its design architecture. It provides a range of graph drawing functionality for client-side or server-side applications. JGraph has a simple, yet powerful API enabling you to

visualize, interact with, automatically layout and perform analysis of graphs”[17]. Support of dragging and cloning cells, re-sizing and re-shaping, connecting and disconnecting facilities of JGraph made the development of CCTRflow graphical designer tool easier and fast.

**InterProlog:** “InterProlog is a Java front-end and enhancement for Prolog. InterProlog is an open source Java front-end and functional enhancement for standard Prologs, running on Windows, Linux and Mac OS X, and currently supporting the top open source logic engines. Mutual recursion and (Java) multithreading are supported. Java Reflection and Serialization mechanisms, together with Prolog’s natural strengths, are used to give the combination great flexibility and dynamism. Rather than tasting like an objectified Prolog/C interfaces, InterProlog provides a higher-level API equating objects to terms, inducing a more concise and declarative programming style”[18]. InterProlog framework provided a flexible communication between Prolog and Java by hiding various details of platform interactions. Interprolog enabled to call any Prolog goal from Java side and to call any java method from prolog side with a single instruction.

**XSB:** “XSB is a Logic Programming and Deductive Database system for Unix and Windows. It is being developed at the Computer Science Department of the Stony Brook University, in collaboration with Katholieke Universiteit Leuven, Universidade Nova de Lisboa, Uppsala Universitet and XSB, Inc.”[15]

**SWI:** “SWI-Prolog offers a comprehensive Free Software Prolog environment, licensed under the Lesser GNU Public License. Together with its graphics toolkit XPCE, its development started in 1987 and has been driven by the needs for real-world applications. These days SWI-Prolog is widely used in research and education as well as for commercial applications.”[16]

## **CHAPTER 4**

### **USER INTERACTION AND GRAPHIC DESIGN**

This chapter presents the graphical design developed for the CCTR Environment. Underlying design decisions are discussed and compared to the criteria given in Chapter 3. Additionally this chapter can be considered as the user's manual of CCTR Environment.

#### **4.1.CCTR Environment Introduction**

A typical user scenario in CCTR environment is as follows:

- CCTR Application is started
- A new CCTRflow diagram is created or a previously saved diagram is opened.
- A CCTRflow is formed according to business requirements using WYSWYG editor.
  - CCTRflow tasks are created and named.
  - CCTRflow constraints are created and named
  - CCTRflow resources are defined
  - Resource assignments for each task is made
  - Cost assignments for each task is made
  - Constraints arguments for each constraint are defined.
- The new CCTRflow is executed.

- Each partial schedule found by CCTR Interpreter and, if any, its resource assignment obeying the specified constraints showed to user.

To illustrate the typical usage of CCTR Environment a subpart of “House Construction Workflow” problem defined in Example 1.1 will be used in the rest of this thesis.

**Example 4.1:** Example 4.1 is subpart of the Example 1.1 that spans **carpentry**, **roof**, **installations** and **gardening** tasks. The task names are abbreviated to **c,r,i,g** respectively for illustrative purposes. The precedence relations: task **c** must be completed before task **r** starts, task **[c,r],[i],[g]** can be done in parallel. There are three machine resources **r1, r2, r3**, each capable of doing **c,r** tasks. Tasks **i,g** can be done only by resources **r1, r2**. All resource task couples have different time, cost, and price value. Values for the example are shown in Table 4.1. The resource allocation and cost constraints are as follows:

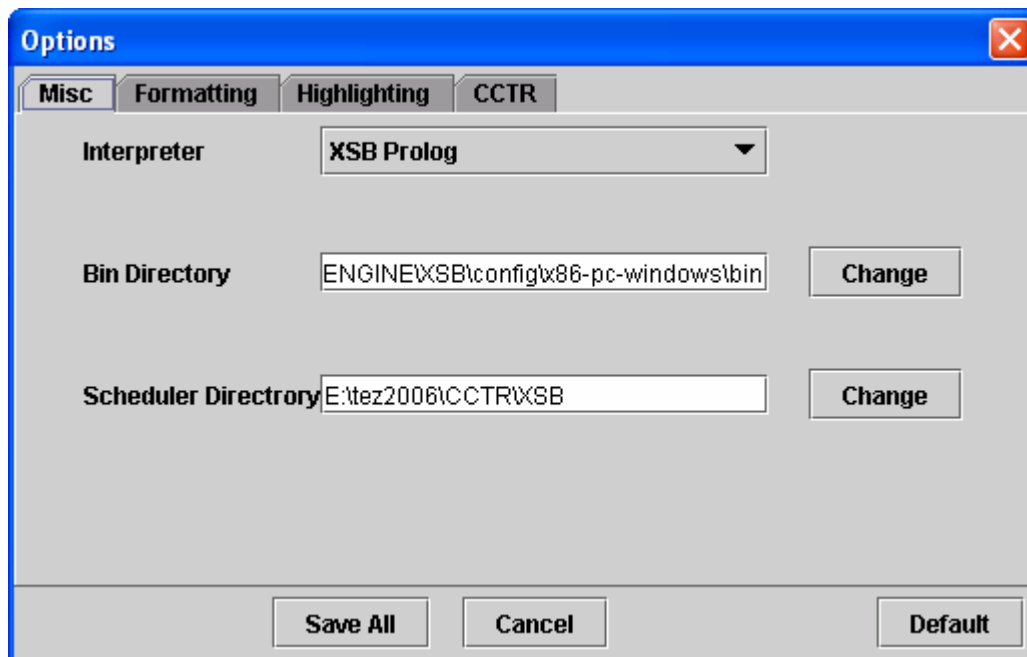
- The total price cost of completing **c,r** tasks should be less than 15000 \$
- The total price cost of the workflow should be less than 30000 \$
- The total duration of the workflow should be less than 10 days.
- The set of resources assigned to concurrent braches should be disjoint.

**Table 4.1:** Example 4.1 Constraints Table.

<b>Task</b>	<b>Resource</b>	<b>Price</b>	<b>Duration</b>
carpentry	R1	1000	1
	R2	3000	2
	R3	8000	3
roof	R1	2000	3
	R2	4000	4
	R3	6000	5
installations	R1	9000	1
	R2	7000	2
gardening	R1	6500	5
	R2	7500	7

## **4.2.Starting CCTR Environment**

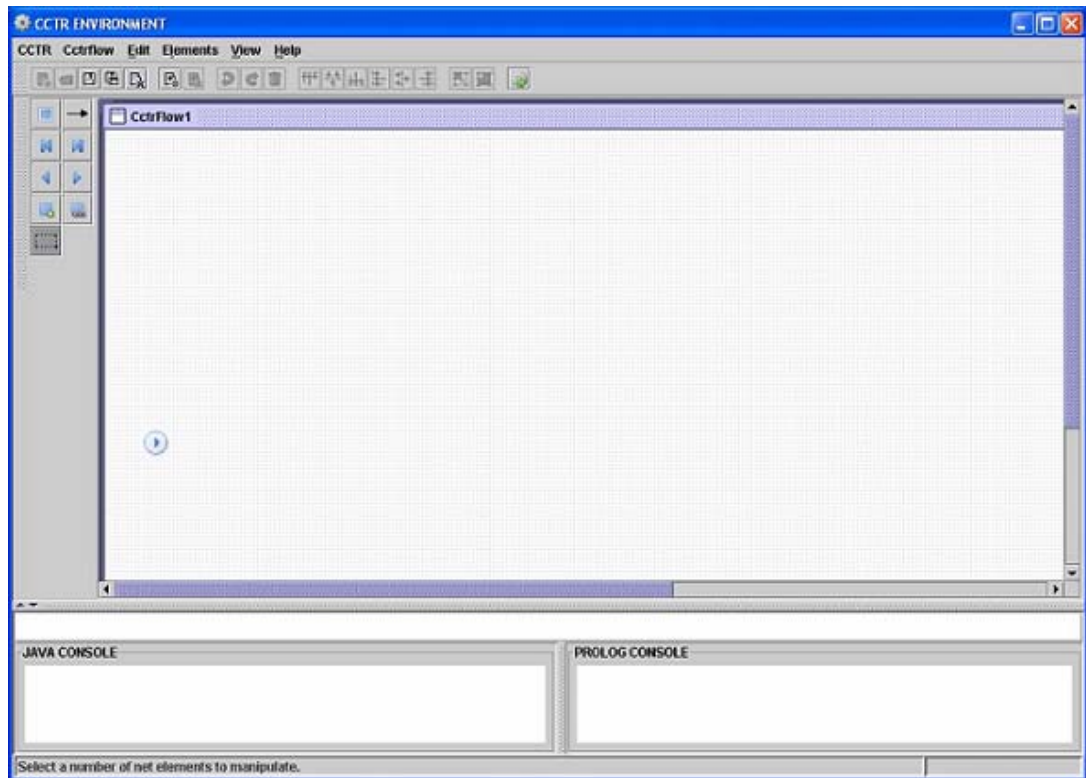
As explained in Chapter 3, CCTR Environment integrates three platforms namely, Java, XSB and SWI. Although, all Java applications are platform independent there is still need to make some configuration in order to integrates these platforms. In CCTR Environment, all configurations related to the application are kept in an XML file. All settings in configuration file can be modified using Configuration Window shown in Figure 4.1 The details about configuration and how to start CCTR Environment are explained in Appendix A. “Tutorial: Starting CCTR Environment”.



**Figure 4.1:** Configuration window.

### 4.3.CCTR Application

All of the user interactions during the realization of the scenario in 4.1 is controlled by main application window showed in Figure 4.2.



**Figure 4.2:** Main window of CCTR Environment.

Main application window consist of a menu bar that holds the menu items for triggering various CCTR actions, a toolbar that holds some of the frequently used menu items, a CCTRflow palette that holds the selection button elements for CCTRflow, a CCTRflow panel that holds the diagram of CCTRflow, a console panel that shows messages created by Java and Prolog , and a status bar that informs user about the state changes in the application.

#### **4.4.Menu Overview**

In this section a brief overview of CCTR menu located on top of CCTR environment will be explained.

CCTR Cctrflow Edit Elements View Help

**Figure 4.3:** Main Menu of CCTR Environment.

**CCTR:** CCTR Menu holds all the standard file options of Create, Open, Save, Close and Exit for CCTRflow design files. It also contains menu items to manage resource and cost definitions, to configure CCTR options and to print CCTRflow diagram.

**CCTRflow:** CCTRflow menu provides options to create, remove, and run the currently selected CCTRflow. It also contains menu items for setting flow resources, showing CCTR syntax, exporting CCTRflow diagram to a image file and showing result window.

**Edit:** Edit Menu provides the standard options of Undo, Redo, Cut, Copy, Paste and Delete objects within CCTRflow diagram.

**Elements:** Elements menu allows user to align CCTRflow elements within the diagram and modify their sizes.

**View:** View menu allows to turn off/on tool tips, grids in diagrams, anti-alias in diagrams.

**Help:** Help menu contains standard help menu items like author, copyright, acknowledges, and help contents.






## 4.5.Creating CCTRflow Diagrams

In this section a step by step guide for creating CCTRflow diagram will be explained. The example defined in 4.1 will be used for the illustration purpose.





**Step 1.** To create a new CCTRflow user selects the *CCTR → Create Design* menu item. Clicking this menu items enables the Pallet Bar and creates an empty canvas for editing CCTRflows. Created canvas contains a *Start* and *Finish* element by default. These default elements can't be removed from any CCTRflow diagram.

**Palette Bar:** Palette Bar holds nine selector buttons that is used with creation, selection and positioning of CCTRflow elements within in the diagram. The Palette bar items are also accessible by right-clicking anywhere on a CCTRflow diagram that does not contain an element. Once an element is selected, it is possible to drop the selected element in the canvas by left- clicking the mouse button. Available CCTRflow elements and their usages are explained in Table 4.1.

**Table 4.2:** CCTRflow palette elements.

ICON	NAME	EXPLANATION
	CCTR Task	<p>Selecting this button creates an atomic task which will be performed by any external factor. This diagram element corresponds to the <i>Atomic Goal</i> in definition 2.3.1</p> <p>Sequence of this diagram element corresponds the <i>Serial Goal</i> of definition 2.3.1</p>
	Parallel(AND) Split	Selecting this button creates a Parallel Split element that shows the start of any parallel branch in a diagram.
	Parallel(AND) Join	<p>Selecting this button creates a Parallel Join element that shows the end of any parallel branch in a diagram.</p> <p>Parallel Split and Parallel Join Elements together corresponds the <i>Parallel Goal</i> of definition 2.3.1</p>
	Disjunctive(OR) Split	Selecting this button creates a disjunctive(OR) Split element that shows the start of any selection branch in a diagram.
	Disjunctive(OR) Join	<p>Selecting this button creates disjunctive(OR) element that shows the end of any selection branch in a diagram.</p> <p>Disjunctive Split and Disjunctive Join Elements together corresponds the <i>Disjunctive Goal</i> of definition 2.3.1</p>

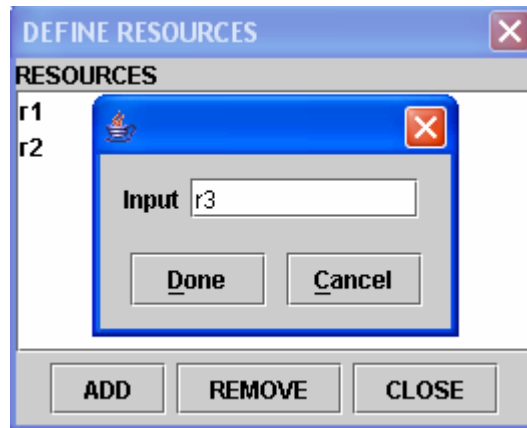
**Table 4.2:** CCTRflow palette elements(Continued)

	Cost Constraint	<p>Selecting this button creates a cost constraint element that enables to define cost constraints in a diagram</p> <p>This diagram element corresponds the <i>Constraint Goal</i> of definition 2.3.1</p>
	Resource Constraint	<p>Selecting this button creates a resource constraint element that enables to define resource constraints in a diagram</p> <p>This diagram element corresponds the <i>Constraint Goal</i> of definition 2.3.1</p>
	Flow Connector	<p>Selecting this button creates flow connector that enables to connect created elements in a diagram.</p>
	Element Selector	<p>Selecting this button enables to select CCTRflow diagram elements for processing(resizing, deleting, moving etc.)</p>

**Step 2.** Second step of creating CCTRflow is dragging and dropping necessary elements from palette bar to the canvas and connecting them using *Flow Connector* according to problem requirements. At this step various alignment properties in the *Elements menu* can be used for arranging CCTRflow diagrams.

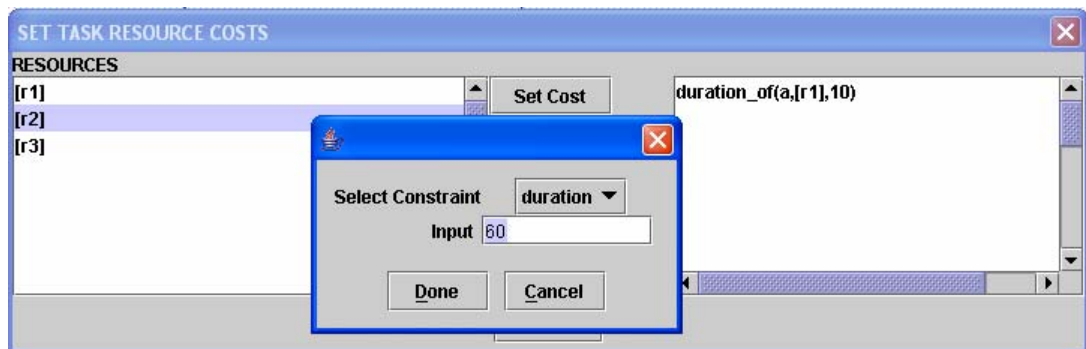
**Step 3.** At the third step, user names CCTR tasks, cost and resource constraints. Any name can be given for CCTR task by simply right clicking specified task and entering the name to the opened input dialog. Unlike CCTR tasks, cost and resource constraints must be selected among the predefined constraints. Constraint definitions are managed via the *CCTR → Update Cost Constraint Definitions* and *CCTR → Update Resource Constraint Definitions* menu items.

**Step 4.** In this step user defines the resources that will be used in current diagram. For this purpose user clicks the *CCTRflow → Define Resources* menu item. Clicking menu item opens the dialog window showed in FigureXX. With define resource window user can add, remove resources for the current CCTRflow.



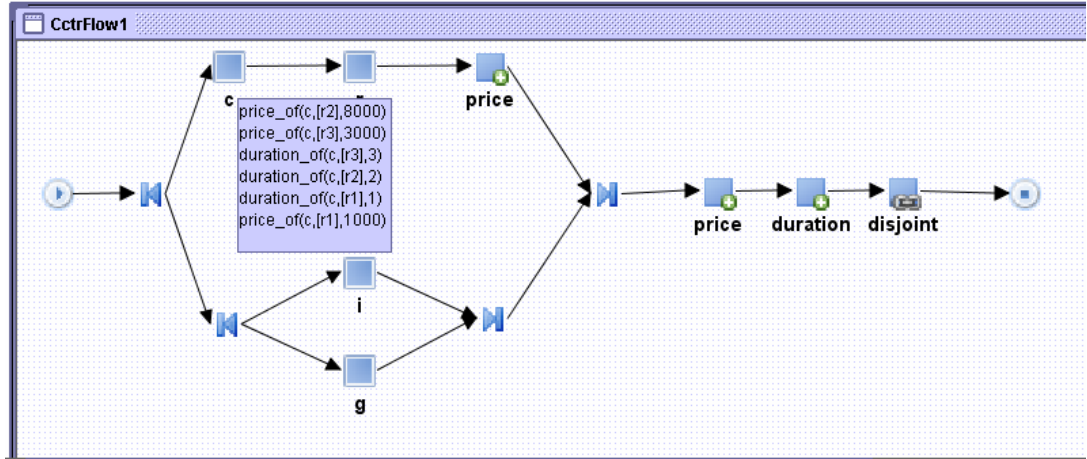
**Figure 4.4:** Define resources window

**Step 5.** In this step user makes the resource and cost assignments for each task and sets the argument for each constraint. For this operation user uses the pop-up menu accessed by right clicking the specified CCTR task element or CCTR Constraint. The window used for resource cost assignment is showed in Figure 4.5.




**Figure 4.5:** Define resource costs window

**Step 6.** For the last step user saves created diagram by simply clicking the *CCTR* → *Save* menu item. Created diagram for “Example 4.1” is showed in Figure 4.6.



**Figure 4.6:** House Construct CCTRflow diagram

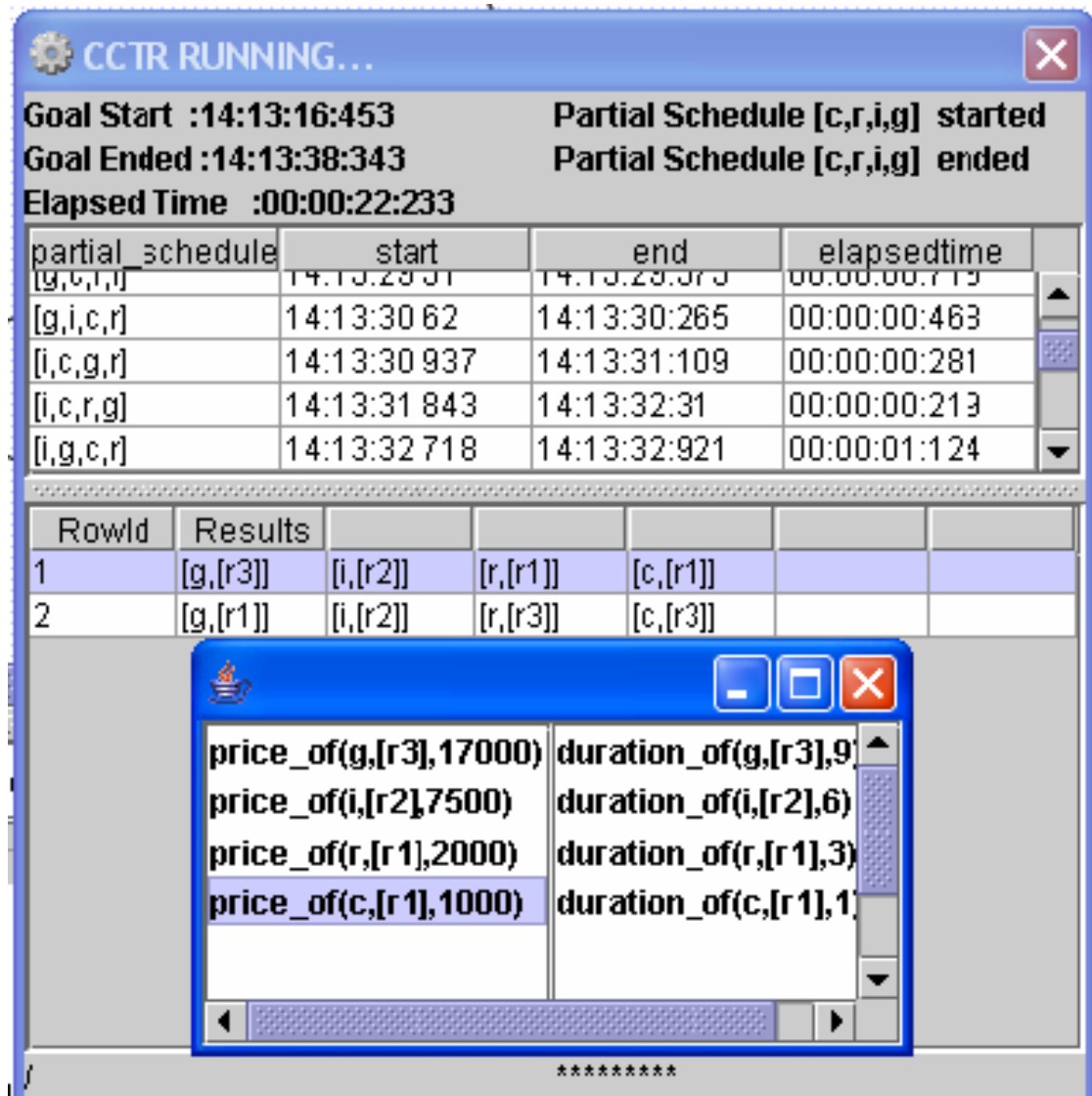
## 4.6. Running CCTRflows

After creating a valid CCTRflow, it can be executed by clicking the *CCTRflow* → *RunCCTR* menu item or clicking the  item in the toolbar. Executing of CCTRflow starts with constructing the CCTR syntax of diagram. Constructed syntax by visiting all the diagram elements recursively is showed to the user on the bottom of the main window. Syntax constructed for Example 4.1 is “(((c\*r)&cost\_price[15000])#(g#i)))&cost\_price[30000]&cost\_duration[10]& rsrc\_disjoint[0]

Another output of visiting all diagram elements is a file that holds all resource and cost predicates.

## 4.7. Showing Results

While executing a CCTRflow, a progress and result window is presented to the user. Window represents the Start and Finish time of CCTR goal, list of all partial schedules found and ,if any, all the resource assignments. User can access the last CCTRflow result window from *CCTRflow* → *Show Result Window*.



**Figure 4.7:** House Construct CCTRflow Results

## CHAPTER 5

### DESIGN AND IMPLEMENTATION

This chapter describes design and implementation of CCTR Environment. As explained in the chapter 3 of this thesis, CCTR Environment integrates three platforms in order to achieve the goals of this thesis. This chapter gives information about details of design and implementation for each platform. Although physically three platform is used for the implementation, conceptually system can be analyzed in two parts, *Application Control and GUI(Java Part)* and *CCTR Scheduler (Prolog Part)*.

#### 5.1.Application Control and GUI(Java Part)








As stated before CCTR Environment is a Java Application that communicates with XSB and SWI. In chapter 4, detail explanation of Graphical User Interface was given. In this section, menu events, windows events and java programming related implementation details are skipped. Since application conforms to the common java development standards any one familiar with java programming can easily figure out it. Instead, data structures used in the development of CCTRflow diagram, and how CCTR formulas are formed from the diagram will be explained.

##### **Forming CCTR Syntax From CCTRFlow Diagram:**

Creating of CCTR syntax from the CCTRflow diagram is achieved simply by visiting the diagram elements recursively and collecting the data associated with them. There are three outputs that span the CCTRFlow diagram; a string that holds CCTRflow in CCTR syntax, a file that holds resource and cost predicates, another file that holds resource and cost constraints definitions.

When generating CCTR Syntax, CCTRFlow diagram element CCTR Syntax equivalences are used. CCTRFlow diagram elements and their equivalences in CCTR goal and CCTR Syntax are explained in Table 5.1.

**Table 5.1:** CCTRflow palette elements.

DIAGRAM ELEMENT	NAME	CCTR GOAL	CCTR SYNTAX
	CCTR Task	<i>Atomic goal:</i>	Task Name
	CCTR Task Sequence	<i>Serial goal:</i> $\phi_1 \otimes \dots \otimes \phi_2$	$(\phi_1 * \dots * \phi_2)$
	Parallel(AND) Split and Join	<i>Parallel goal :</i> $\phi_1 \mid \dots \mid \phi_2$	$(\phi_1 \# \dots \# \phi_2)$
	Disjunctive(OR) Split and Join	<i>Disjunctive goal:</i> $\phi_1 \vee \dots \vee \phi_2$	$(\phi_1 \vee \dots \vee \phi_2)$
	Cost Constraint	<i>Constraint goal :</i> $\phi_1 \wedge \text{constr}$	$\wedge \text{cost\_ConstraintName}$
	Resource Constraint	<i>Constraint goal :</i> $\phi_1 \wedge \text{constr}$	$\wedge \text{rsrc\_ConstraintName}$
	Start, Finish	-	$(\dots\dots)$

In order to create a valid CCTR Syntax from a CCTRflow following properties should be associated for CCTR Task elements and Constraint Elements.

#### CCTR Task:

*Name:* Name of Task

*Resources Assignments:* Every task holds the data of resource assignments associated with it.

*Cost Assignments:* Every task holds the data of cost assignments associated with it.

### **CCTR Constraint:**

*Name:* Name of the Resource or Cost Constraint.

*Arguments:* Arguments of the Resource or Cost Constraint

Generated CCTR Syntax for Example 4.1 is:

```
(((((c*r)&cost_price[15000])#(g#i)))&cost_price[30000]&cost_duration[10]&rsrc_disjoint[0]
```

Second output of spanning CCTRflow diagram is a file that contains all the resource and cost assignments. When traversing CCTRflow for every resource assignment of a task predicate **resource(Task,Resource)** is created. Similarly, for every cost assignment of a task predicate **constraintName\_of(Task,Resource,Cost)** is created.

Third output is the file that contains predefined constraint definitions. It contains a predicate for each constraint in CCTRflow diagram. Definitions of constraint are created by using the templates explained in 2.3.4

## **5.2.CCTR Scheduler(Prolog Part)**

After handling all the user interactions, Java Part, generates three input for CCTR Scheduler; A string that holds CCTRflow in CCTR syntax, a file that holds resource and cost predicates, another file that holds resource and cost constraints definitions. The details of these inputs were explained in 5.1.

To pass the control of the system to the Prolog Part, Java Part first loads the *CCTRload.P* prolog file and then calls the **cctr** goal in it by using the *deterministic Goal* method of *XSBPrologEngine* class. Calling **cctr** goal starts the execution of CCTR Scheduler which consist of four subsystem, CCTRLoader, Transformer, CTR Interpreter, and Constraint Solver.

**CCTRLoader:** CCTRLoader subsystem is responsible for preparing scheduler (compiling, consulting prolog files etc.) and controlling the flow of the application in prolog side. Entry point in CCTRLoader is the *cctr(WorkFlow,CctrFilename,IsAll)* predicate.

*WorkFlow*: String presentation of workflow to be scheduled in CCTR syntax.

*CctrFilename*: File with a .cctr extension that holds all the resource and cost predicates.

*IsAll*: Boolean flag for controlling partial schedules. If this flag is true all possible partial schedules are checked for a valid schedule otherwise execution stops when a valid schedule with resource assignments is found.

An example of calling *cctr* predicates for the Example 4.1 is as follows.

### Example 5.2.1

```
cctr('((((c*r)&cost_price[15000])#(g#i)))&cost_price[30000]&cost_duration[10]&
rsrc_disjoint[0]','CCTR_HOME/example14.cctr',true).
```

*Initializing the CCTR Scheduler*: Initializing starts with compiling and consulting the source files. All the CCTR Scheduler source files resides in CCTR\_HOME/XSB/ directory. The content of this directory is as follow.

- *CctrLoad.P*: Predicates for starting and controlling CCTR Scheduler.
- *Transformation.P*: Predicates that carry out the transformation explained in 2.3
- *Upload.P*: A module that loads the CTR transactions base into the XSB[3]
- *Parser.P*: A parser for CTR rules[3]
- *Ctr.P*: The basic CTR Interpreter and code for backtrackable updates. [3]
- *Constraints.P*: Holds the dynamically created resource and cost definition predicates that generates constraint set for the executing scheduler.

One of the design goals of the CCTR Scheduler is reusing the CTR implementation in [3] as much as possible. For this purpose, CCTRLoad and Transformer subsystems makes some processing on CCTR goal and input file that contains resource and cost constraints and generates a transaction base file which is acceptable by CTR Interpreter.

**Transformer:** Transformation algorithm given in 2.3.3 is implemented in *Transmformer.P* module. Transformation is implemented as Definite Clause Grammar(DCG) in XSB. Since it is easy to apply transformation algorithm extensions, writing a context free grammar preferred instead of string processing. Grammar source is presented in Appendix 2. For the implementation purposes the algorithm given in 2.3.3 is slightly modified. Modified algorithm is given in Table 5.2. When transforming CCTR goal, a list of task, Resource Variable is also created. Purpose of this list is to pass the result of CCTR scheduler or resource assignments to Java Part.

**Table 5.2:** Modified Transformation Rules.

	$\Theta(G): \rightarrow \Theta_1(G)$
	<b>Transformation <math>\Theta_1</math>;</b> uses auxiliary transformation $\Theta_2$
(1)	$\Theta_1(A): \rightarrow A$ , if A is a an atomic formula
(2)	$\Theta_1(G \wedge \text{constr}): \rightarrow \Theta_1(G) \otimes \underline{\text{constr}}(\Theta_2(G))$ , $\text{constr} \in C$ is constraint predicate and $\underline{\text{constr}}$ is a new predicate that simulates $C_D$ - interpretation of c in constraint universe D.
(3)	$\Theta_1(G_1 \text{ op } G_2): \rightarrow \Theta_1(G_1) \text{ op } \Theta_1(G_2)$ , if op is $ $ , $\otimes$ , or $\vee$
	<b>Auxiliary transformation <math>\Theta_2</math>;</b>
(1)	$\Theta_2(A): \rightarrow \text{const}(A, \text{Var}_A)$ , if A is an atomic formula const is a constant predicate used by implementation. $\text{Var}_A$ is a new variable that represents the resource assignment of A.
(2)	$\Theta_2(G \wedge \text{constr}): \rightarrow \Theta_2(G)$ , where $\text{constr} \in C$ is a constraint predicate

**Table 5.2:** Modified Transformation Rules(Continued).

(3)	$\Theta_2(G_1 \otimes G_2) : \rightarrow \bullet_p(\Theta_2(G_1), \Theta_2(G_2))$
(4)	$\Theta_2(G_1 \mid G_2) : \rightarrow \parallel_p(\Theta_2(G_1), \Theta_2(G_2))$
(5)	$\Theta_2(G_1 \vee G_2) : \rightarrow \gamma_p(\Theta_2(G_1), \Theta_2(G_2))$

Transformation process starts with calling `cctr_trans(WorkFlow,ResultVar,Out)` predicate.

*WorkFlow*: String presentation of workflow to be scheduled in CCTR syntax.

*ResultVar*: Variable which will hold the Task, Resource Variable pair list. This List will be used to pass CCTR Scheduler to Java Part. In the implementation ‘DummyVariable’ name is used.

*Out*: Transformed CCTR Syntax.

Transformed syntax for the example workflow syntax in 5.2.1 is as follows

Out: (((((g#i)#((c\*r)))))))\* (

DummyVariable=[[g,'G'],[i,'I'],[r,'R'],[c,'C']],

Constraints=[

cost\_price([15000],@(const(c,'C'),const(r,'R'))),

cost\_price([30000],\$(const(g,'G'),const(i,'I')),@(const(c,'C'),const(r,'R'))),

cost\_duration([10],\$(const(g,'G'),const(i,'I')),@(const(c,'C'),const(r,'R'))),

rsrc\_disjoint([0],\$(const(g,'G'),const(i,'I')),@(const(c,'C'),const(r,'R')))],

compile\_constraints(Constraints,FileName,DummyVariable),execute\_constraints(FileName)).

Transformer uses following syntax instead of the predicates introduced in 2.3.3

**Table 5.3:** Transformer partial predicates equivalences .

Serial Partial	$\bullet_p \rightarrow @$
Parallel Partial	$\parallel_p \rightarrow \$$
Disjunctive Partial	$\gamma_p \rightarrow ^\wedge$

**CTR Interpreter:** The CTR Interpreter implementation developed in [3] is used for CCTR Scheduler. Only modification to this implementation is addition of **schedule(Task)** predicate and **onBacktrackingSchedule(Task)** predicates. These two predicates are needed for two purposes. First one is to provide found partial schedules to GUI part during the execution. Second one is to handle disjunctive goals in CCTR formulas. In [8], CCTR formulas are first transformed to Disjunctive Normal Form(DNF) before applying the Transformation rules in Table 2.1. DNF transformation phase role is to eliminate disjunctions from constraints definitions. In this implementation instead of using DNF, **partial\_schedule(Task)** predicate is used in the constraint generation step in order to test whether a task in disjunctive goal is in partial schedule or not. Algorithm is as follows:

**schedule(Task)** is called whenever a task called or committed. It appends the **Task** into the maintained partial schedule list and asserts the **partial\_schedule(Task)** predicate.

**onBacktrackingSchedule(Task)** is called whenever a task is backtracked or roll backed. It reverses operation of **schedule(Task)** predicate. It removes the **Task** from the maintained partial schedule list and retracts **partial\_schedule(Task)** predicate.

**Constraint Solver:** Constraint solver mechanism is as follows. Each time CTR Interpreter found a partial schedule, a constraint set file with extension .cst is created. Constraint file contains all resource and cost predicates and a new predicate **executeConstraints([PartialSchedule])**. **executeConstraints** predicate body is created dynamically by CTR Interpreter using the constraint definitions of workflow. **executeConstraints([PartialSchedule])** finds all of the resource assignments for the current schedule.

For the implementation of constraint solver **SWI**[16] **clp/bounds** library is used. The bounds solver is a rather simple integer constraint solver, implemented with attributed variables. Its syntax is a subset of the SICStus[21] **clp(FD)** syntax. **SWI clp/bounds** library suited well for our examples.

## **CHAPTER 6**

### **CONCLUSIONS**

In this thesis, we implemented a logical framework proposed for scheduling workflows under resource allocation constraints and developed an extensible and flexible graphical user interface for the proposed framework. Formalization of this framework developed in [8] is called Concurrent Constraint Transaction Logic that integrates Concurrent Transaction Logic [4] with Constraint Logic Programming [20]. Implementation of semantic formalism and provided algorithm resulted in correct solutions for workflow scheduling under resource allocation constraints. The developed system, which integrates Prolog and Java Platforms, is designed to serve as the basis for many application domains that involves CCTR based workflows and schedulers. For example, implemented CCTR Scheduler and developed CCTR Environment can be extended to use by workflow scheduling, resource management in workflows, scheduling and planning, and agent-based workflow systems. CCTR Scheduler can also be used for composing Web services in which Web Services are modeled as resources in conjunction with a planning system [8]

As a future work developed system can be extended with special-purpose constraint solvers optimized for workflow scheduling. For instance, Constraint handling rules (CHR)[21] which is a concurrent committed-choice constraint programming language can be used to extend Constraint Solver part of the implementation.

## REFERENCES

1. A.J. Bonner and M. Kifer. Transaction logic programming. Technical Report CSRI-323, Computer Systems Research Institute, University of Toronto, April, 1995.
2. Samuel Y.K Hung . Implementation and performance of transaction logic in Prolog. Degree of Master Science, Department of Computer Science , University of Toronto, 1996
3. Amalia F. Sleghel . An optimizing interpreter for concurrent transaction logic. Degree of Master Science, Department of Computer Science , University of Toronto, 2000
4. A.J. Bonner and M. Kifer. Concurrency and communication in transaction logic. In Joint Int'l Conference and Symposium on Logic Programming, pages 142–156, Bonn, Germany, September 1996, MIT Press.
5. A.J. Bonner and M. Kifer. Transaction logic programming. In Int'l Conference on Logic Programming, pages 257–282, Budapest, Hungary, June 1993. MIT Press.
6. A.J. Bonner. Workflow, transactions, and datalog. In ACM Symposium on Principles of Database Systems, Philadelphia, PA, May/June 1999.
7. H. Davulcu, M. Kifer, and I.V. Ramakrishnan. Ctr-s: A logic for specifying contracts in semantic web services. In 13th International World Wide Web Conference (WWW2004), May 2004.
8. P.Senkul, M.Kifer, and I.H. Toroslu. Logic based-modeling and scheduling of workflows under resource and cost constraints. , September 2005.
9. Special issue on workflow systems. Bulletin of the Technical Committee on Data Engineering (IEEE Computer Society), 18(1), March 1995.
10. D. Georgakopoulos, M. Hornick, and A. Sheth. An overview of workflow management:From process modeling to infrastructure for automation. Journal on Distributed and Parallel Database Systems, 3(2):119–153, April 1995.
11. G. Alonso, D. Agrawal, A. El Abbadi, and C. Mohan. Functionality and Limitations of Current Workflow Management Systems. In IEEE-Expert. Special issue on Cooperative Information Systems, 1997

12. F.Wan, K.Rustogi, J.Xing, and M.P. Singh. Multiagent workflow management. In IJCAI Workshop on Intelligent Workflow and Process Management: The New Frontier for AI in Business, Stockholm, Sweden, August 1999.
13. C. Schulte and G. Smolka. Finite Domain Constraint Programming in Oz. A Tutorial. Version 1.1.0, February 2000.
14. Java Official Web Site <http://java.sun.com>, August 2006
15. XSB Web Site <http://xsb.sourceforge.net/>, August 2006
16. SWI Web Site <http://www.swi-prolog.org/>, August 2006
17. JGraph Web Site <http://jgraph.sourceforge.net> , August 2006
18. InterProlog <http://www.declarativa.com/InterProlog/default.htm> , August 2006
19. A.J. Bonner. Transaction Datalog: a compositional language for transaction programming. In Proceeding of the International Workshop on Database Programming Languages, number 1369 in LNCS, pages 373-395. Springer-Verlag, 1998. Workshop
20. J.Jaffar and J.-L Lassez. Constraint logic programming: A survey . Journal of Logic Programming, 19-20 May 1994
21. T.Frihwirth. Programming in Constraint Handling Rules, [www.informatik.uni-ulm.de/pm/mitarbeiter/fruehwirth/](http://www.informatik.uni-ulm.de/pm/mitarbeiter/fruehwirth/) , August 2006
22. SICStus prolog. <http://www.sics.se/isl/sictus.html> , August 2006

## APPENDIX A

### TUTORIAL : STARTING CCTR ENVIRONMENT

The following is a tutorial on how to get started using **CCTR Environment**. This tutorial assumes that Windows or Linux operating systems are used. The instructions are analogous for other operating systems as well

1. **Installing Java: CCTR Environment** requires that the Java Software Developers Kit JDK 1.4 or later be installed. If JDK 1.4 does not exist in your computer download and install it from <http://java.sun.com/j2se/1.4.2/download.html>.
2. **Installing XSB:** CCTR Scheduler part of CCTR Environment requires that XSB Version 2.7.1 or later be installed. XSB can be downloaded and installed from <http://xsb.sourceforge.net>
3. **Installing SWI:** Constraint solver part of CCTR Environment requires that SWI Prolog version 5.6.1 or later be installed. SWI prolog can be download and installed from <http://www.swi-prolog.org>
4. **Installing CCTR Environment:** Download the *cctr{version}.zip* file, and expand it into somewhere in your disk . Assuming CCTR is installed to the CCTR\_HOME directory. Content of the CCTR\_HOME directory is as follow.
  - */dist:* Directory containing *cctr{version}.zip* file
  - */build:* Directory containing the build files.
  - */docs:* Directory containing documentation.

- */lib*: Directory containing dependent jar files; *interprolog.jar*, *jgraph.jar*, *swing-layout1-0.jar*
- */src*: Directory containing java source package and files. It also contains ant **build.xml** and **default.properties** files for re-building and packaging the project.
- */xsbsource*: Directory containing the CCTR Scheduler implementation source files.
- */scripts*: Directory containing the scripts for running CCTR Environment in Windows and Linux operating systems.
- */examples*: Directory containing some example CCTRflow diagrams.
- *cctr{version}.jar*: Runnable java jar file for CCTR Environment.
- *cctrConfig.xml*: XML file containing various settings of CCTR Environment.

5. **Running CCTR Environment:** In order to run CCTR Environment it is needed to execute *runWindowsCCTR.bat* or *runLinuxCCTR.sh* in */scripts* directory. Alternatively *java -jar cctr{version}.jar* command can be invoked when the current directory is CCTR\_HOME.

6. **Mounting Bin Directories:** XSB\_BIN\_DIRECTORY and SWI\_BIN\_DIRECTORY are two directories that should be set for CCTR Environment. There are two ways to set Bin directories. It can be set by simply editing the *cctrConfig.xml* and changing the *INTERPRETER* element *BinPath* properties or using the Config Window accessed from the *CCTR→CCTR Options* menu item.

7. **Re-building Project:** To re-build the entire CCTR Environment project an ant *build.xml* file is provided in */src* directory. If ant tool is not exist in your computer it can be download and installed from <http://ant.apache.org/> . Properties used in the building process are stored in *default.properties* file

Main ant targets are as follow:

- *clean*: Removes all of the automatically generated files.
- *compile*: Compiles a version of the project to the *build/classes* directory
- *jar*: Generates a jar (library of compiled code) for the project and puts it in the top level file *cctr{version}.jar* file.
- *package*: Creates the package file for distribution. Two files are created in */dist* directory; *cctr{version}.zip* and *cctr{version}.tar*

## APPENDIX B

### TRANSFORMER GRAMMER

```
/* Constraint predicates also includes tasks*/
/*Tabling is required*/
:- table cctr_goal/6.
:- table cctr_goal2/3.
:- import concat_atom/2 from string.
:- import append/3 from basics.
:- import member/2 from basics.

cctr_trans(In,PairVar,Out) :-assert('empty'),atom_chars(In,R),
(cctr_goal(Val,Pair,T_Const,T,R,[])->true;write('Unable to parse Cctr WorkFlow'),write(R),fail),

concat_atom_comma(Pair,PairComma),concat_atom(['(',T,')']*('PairVar','=',PairComma,')',',',Constraint
s=['T_Const,'],compile_constraints(Constraints,FileName,'PairVar,'),execute_constraints(FileName)
),',',Out)

.

/*First Part Transformation */

cctr_goal(Val,Pair,T_Const,T)
cctr_goal(Val1,Pair1,T_Const1,T1),ctr_op(Val2),cctr_goal(Val3,Pair2,T_Const2,T2),
{concat_atom([Val1,Val2,Val3],Val),concat_atom([T1,Val2,T2],T),
strong_append(Pair1,Pair2,Pair),(T_Const1
=='empty',T_Const2=='empty'->T_Const='empty';((T_Const1\=='empty',T_Const2\=='empty'-
>concat_atom([T_Const1,',',T_Const2],T_Const);(T_Const1=='empty'-
>T_Const=T_Const2;T_Const=T_Const1))))}.

cctr_goal(Val,Pair,T_Const,T) --> ['(',cctr_goal(Val1,Pair,T_Const,T1),[')'],
{concat_atom(['(',Val1,')'],Val),concat_atom(['(',T1,')'],T)}.

cctr_goal(Val,Pair,T_Const,T) --> ['o','1','('],cctr_goal(Val1,Pair,T_Const,T1),[')'],
{concat_atom(['(',Val1,')'],Val),concat_atom(['o1(',T1,')'],T)}.

cctr_goal(Val,Pair,T_Const,T)
cctr_goal(Val1,Pair,T_Const1,T1),const_op(Val2),const_token(Val1,Val3,T_Const2),
```

```

{concat_atom([Val1,Val2,Val3],Val),T=T1,(T_Const1=='empty'-
>T_Const=T_Const2;concat_atom([T_Const1,',',T_Const2],T_Const))}.

ctr_goal(Val,Pair,T_Const,T) --> ctr_goal_token(Val,Pair,T),{T_Const='empty'}.

ctr_goal_token(Val,Pair,T) --> ['(',ctr_goal_token(Val1,Pair,T1),[')'],
{!,concat_atom(['(',Val1,')'],Val),concat_atom(['(',T1,')'],T)}.

ctr_goal_token(Val,Pair,T) --> res_token(Val1,Up11),ctr_op(Val2),res_token(Val3,Up22),
{!,
concat_atom([Val1,Val2,Val3],Val),concat_atom([Val1,Val2,Val3],T),
concat_atom(['[',Val1,',',Up11,']'],FirstPair),concat_atom(['[',Val3,',',Up22,']'],SecondPair),

add_to_list([],FirstPair,TempPair),add_to_list(TempPair,SecondPair,Pair)}.

ctr_goal_token(Val,Pair,T) --> res_token(Val,Up),{!,concat_atom([Val],T),
concat_atom(['[',Val,',',Up,']'],FirstPair),add_to_list([],FirstPair,Pair)}.

const_token(Alfa,Val,T) --> const_r_token(Val1,T1),const_op(Val2),const_token(Alfa,Val3,T2),
{!,concat_atom([Val1,Val2,Val3],Val),
trans2(Alfa,T3),concat_atom([T1,T3,']'],T4),
concat_atom([T4,',',T2],T)}.

const_token(Alfa,Val,T) --> const_c_token(Val1,T1),const_op(Val2),const_token(Alfa,Val3,T2),
{!,concat_atom([Val1,Val2,Val3],Val),
trans2(Alfa,T3),concat_atom([T1,T3,']'],T4),
concat_atom([T4,',',T2],T)}.

const_token(Alfa,Val,T) --> const_r_token(Val,T1),{!,trans2(Alfa,T2),concat_atom([T1,T2,']'],T)}.
const_token(Alfa,Val,T) --> const_c_token(Val,T1),{!,trans2(Alfa,T2),concat_atom([T1,T2,']'],T)}.

const_r_token(Val,T) --> fix_r_const(Val1),res_token(Val2,Up),cost_args(Val3),
{!,concat_atom([Val1,Val2,Val3],Val),concat_atom([Val1,Val2,('Val3,',',',T)}).

const_c_token(Val,T) --> fix_c_const(Val1),res_token(Val2,Up),cost_args(Val3),
{!,concat_atom([Val1,Val2,Val3],Val),
concat_atom([Val1,Val2,('Val3,',',',T)}).

/*Second Part Transformation*/

trans2(Alfa,Res) :- true,!,atom_chars(Alfa,Out),ctr_goal2(Res,Out,[]).

ctr_goal2(T) --> ctr_goal2(T1),ctr_op2(Val),ctr_goal2(T2),
{concat_atom([Val,('T1,',',',T2,')'],T)}.

ctr_goal2(T) --> ['(',ctr_goal2(T),[')']].

```

```

ctr_goal2(T) --> ctr_goal2(T),const_op(Val),const_token2.
ctr_goal2(T) --> ctr_goal_token2(T).

ctr_goal_token2(T) --> ['(]',ctr_goal_token2(T),[')'],
                        {}.

ctr_goal_token2(T) --> res_token(Val1,Up11),ctr_op2(Val2),res_token(Val3,Up22),
                        {!,

                        concat_atom([Val2,'(const(',Val1,',',Up11,'"'),const(',Val3,',',Up22,'"')')',T)}.

ctr_goal_token2(T) --> res_token(Val,Up1),{!,concat_atom(['const(',Val,',',Up1,'"')',T)}).

const_token2 --> const_r_token2,const_op(Val),const_token2,{}.
const_token2 --> const_c_token2,const_op(Val),const_token2,{}.
const_token2 --> const_r_token2,{}.
const_token2 --> const_c_token2,{}.

const_r_token2 --> fix_r_const(Val1),res_token(Val2,Up),cost_args(Val3),{}.
const_c_token2 --> fix_c_const(Val1),res_token(Val2,Up),cost_args(Val3),{}.

/*Part Used by Two Transformations*/
res_token(Val,Up) --> word(Val1,Up1),res_token(Val2,Up2),
                        {!,concat_atom([Val1,Val2],Val),concat_atom([Up1,Up2],Up)}.

res_token(Val,Up) --> word(Val,Up),{!}.

fix_r_const(Val) --> ['r','s','r','c','_'],{!,Val=rsrc_}.
fix_c_const(Val) --> ['c','o','s','t','_'],{!,Val=cost_}.

cost_args(Val) --> ['[',cost_args_base(Val1),[']'],{!,concat_atom(['[',Val1,']'],Val)}.
cost_args(Val) --> ['[',[']'],{!,Val=[0]}.
cost_args_base(Val) --> int(Val1),[','],cost_args_base(Val2),{!,concat_atom([Val1,',',Val2],Val)}.
cost_args_base(Val) --> int(Val).

int(Val) --> ['-'],intpos(Val1),{!,concat_atom(['-',Val1],Val)}.
int(Val) --> intpos(Val),{!}.

intpos(Val) --> digit(Val1),intpos(Val2),{!,concat_atom([Val1,Val2],Val)}.

```

intpos(Val) --> digit(Val),{!}.

ctr\_op(Val) --> ['\*'],{!,Val='\*'}

ctr\_op(Val) --> ['#'],{!,Val='#'}

ctr\_op(Val) --> ['\','/'],{!,Val='\','/'}

ctr\_op2(Val) --> ['\*'],{!,Val='@'}

ctr\_op2(Val) --> ['#'],{!,Val='\$'}

ctr\_op2(Val) --> ['\','/'],{!,Val='^'}

const\_op(Val) --> ['&'],{!,Val='&'}

digit(Val) --> ['0'],{!,Val=0}

digit(Val) --> ['1'],{!,Val=1}

digit(Val) --> ['2'],{!,Val=2}

digit(Val) --> ['3'],{!,Val=3}

digit(Val) --> ['4'],{!,Val=4}

digit(Val) --> ['5'],{!,Val=5}

digit(Val) --> ['6'],{!,Val=6}

digit(Val) --> ['7'],{!,Val=7}

digit(Val) --> ['8'],{!,Val=8}

digit(Val) --> ['9'],{!,Val=9}

word(Val,Up) --> ['a'],{!,Val=a,Up='A'}

word(Val,Up) --> ['b'],{!,Val=b,Up='B'}

word(Val,Up) --> ['c'],{!,Val=c,Up='C'}

word(Val,Up) --> ['d'],{!,Val=d,Up='D'}

word(Val,Up) --> ['e'],{!,Val=e,Up='E'}

word(Val,Up) --> ['f'],{!,Val=f,Up='F'}

word(Val,Up) --> ['g'],{!,Val=g,Up='G'}

word(Val,Up) --> ['h'],{!,Val=h,Up='H'}

word(Val,Up) --> ['i'],{!,Val=i,Up='I'}

word(Val,Up) --> ['j'],{!,Val=j,Up='J'}

word(Val,Up) --> ['k'],{!,Val=k,Up='K'}

word(Val,Up) --> ['l'],{!,Val=l,Up='L'}

```

word(Val,Up) --> ['m'],{!,Val=m,Up='M'}.
word(Val,Up) --> ['n'],{!,Val=n,Up='N'}.
word(Val,Up) --> ['o'],{!,Val=o,Up='O'}.
word(Val,Up) --> ['p'],{!,Val=p,Up='P'}.
word(Val,Up) --> ['r'],{!,Val=r,Up='R'}.
word(Val,Up) --> ['s'],{!,Val=s,Up='S'}.
word(Val,Up) --> ['t'],{!,Val=t,Up='T'}.
word(Val,Up) --> ['u'],{!,Val=u,Up='U'}.
word(Val,Up) --> ['v'],{!,Val=v,Up='V'}.
word(Val,Up) --> ['w'],{!,Val=w,Up='W'}.
word(Val,Up) --> ['x'],{!,Val=x,Up='X'}.
word(Val,Up) --> ['y'],{!,Val=y,Up='Y'}.
word(Val,Up) --> ['z'],{!,Val=z,Up='Z'}.

%
%auxlary predicates

% A strong append to list Insert Elements to list if they don't exist
strong_append(List1,[Head|Tail],Result) :-
strong_append(List1,Tail,TempResult),add_to_list(TempResult,Head,Result).

strong_append(List1,[],Result) :- Result=List1,!,true.

% A strong insert to a list : Insert Element to list if it doesn't exists.
add_to_list(List,Element,NewList):- not(member(Element,List)),!,append(List,[Element],NewList).
add_to_list(List,Element,NewList):- NewList=List,!,true.

%
concat_atom_comma([Head|Tail],CommaList) :-
concat_atom_comma2(Tail,CommaList1),concat_atom([Head,',',CommaList1],CommaList),true.

concat_atom_comma([Element],CommaList) :- concat_atom([Element],CommaList),true.

concat_atom_comma2([Head|Tail],CommaList) :- concat_atom_comma2(Tail,CommaList1),
concat_atom([Head,',',CommaList1],CommaList).

concat_atom_comma2([Element],CommaList):- concat_atom([Element],CommaList),true.

```

## APPENDIX C

### SAMPLE WORKFLOWS

In this part some sample constraint workflows and their solutions are presented

#### SAMPLE 1.

*CCTR Syntax:*

$(((((f\#e\#d))\&cost\_duration[110])\vee(((h\vee g))\&cost\_duration[85])\vee((a*b*c)\&cost\_duration[70])))*i)\&cost\_duration[250]\&rsrc\_disjoint[0]$

*CONSTRAINTS ASSIGNMENTS:*

**Table 8.1:** Sample 1 Constraints Table.

Task	Resource	Duration
a	r1	10
a	r2	20
b	r1	100
b	r2	20
b	r3	40
c	r1	12
c	r2	150
c	r3	25
d	r1	10

**Table 8.1:** Sample 1 Constraints Table.(Continued)

d	r3	50
d	r4	60
e	r1	60
e	r2	20
e	r3	15
f	r1	150
f	r2	175
f	r3	15
g	r1	75
g	r2	40
i	r3	200
i	r4	50

***SOLUTIONS:*****Table 8.2:** Sample 1 Solution Table.

<b>Flow:</b>	f,e,d,i		
[f,[r3]]	[e,[r1]]	[d,[r4]]	[i,[r4]]
[f,[r3]]	[e,[r2]]	[d,[r1]]	[i,[r3]]
[f,[r3]]	[e,[r2]]	[d,[r1]]	[i,[r4]]
[f,[r3]]	[e,[r2]]	[d,[r4]]	[i,[r4]]
<b>Flow:</b>	a,b,c,i		
[a,[r1]]	[b,[r2]]	[c,[r1]]	[i,[r3]]
[a,[r1]]	[b,[r2]]	[c,[r1]]	[i,[r4]]
[a,[r1]]	[b,[r2]]	[c,[r3]]	[i,[r4]]
[a,[r1]]	[b,[r3]]	[c,[r1]]	[i,[r4]]
[a,[r2]]	[b,[r2]]	[c,[r1]]	[i,[r4]]
[a,[r2]]	[b,[r2]]	[c,[r3]]	[i,[r4]]
<b>Flow:</b>	h,i		
[h,[r4]]	[i,[r3]]		
[h,[r4]]	[i,[r4]]		
<b>Flow:</b>	g,i		
[g,[r1]]	[i,[r4]]		
[g,[r2]]	[i,[r3]]		
[g,[r2]]	[i,[r4]]		

## SAMPLE 2.

### *CCTR Syntax:*

*(((c\*r)&cost\_price[15000])#(g#i)))&cost\_price[30000]&cost\_duration[10]& rsrc\_disjoint[0]*

### **CONSTRAINTS ASSIGNMENTS:**

**Table 8.3:** Sample 2 Constraints Table.

Task	Resource	Price	Duration
C	r1	1000	1
C	r2	3000	2
C	r3	8000	3
R	r1	2000	3
R	r2	4000	4
R	r3	6000	5
I	r1	9000	1
I	r2	7000	2
G	r1	6500	5
G	r2	7500	7

### **SOLUTIONS:**

**Table 8.4:** Sample 2 Solution Table.

Flow:	c,g,i,r		
[c,[r3]]	[g,[r1]]	[i,[r2]]	[r,[r3]]