Building Industrial CHIP Applications from Reusable Software Components

Philip Kay
Helmut Simonis

COSYTEC SA
4, rue Jean Rostand
Parc Club Orsay Université
F-91893 Orsay Cedex
France
{kay, simonis}@cosytec.fr

1. Abstract

In this paper we present results on the systematic reuse of components in a number of large scale applications. The applications have been developed in CHIP, a Prolog based constraint logic programming system. We show that even though the applications solve very different problems, we can reuse large parts of the data model and graphical user interfaces. We describe several very high level components for the handling of textual information in lists, Gantt charts and diagrams. Together with an abstract data model these components significantly decrease development effort required for end-user applications and simplify the extension of existing programs.

2. Introduction

Constraint logic programming (CLP) [JM94] [FHK92] [BC93] has been introduced to simplify the development of complex, decision support systems. It has been shown on many examples that the use of CLP decreases the development time for complex problem solvers and simplifies their maintenance. CHIP (Constraint Handling in Prolog) [DHS88] [VSD92] [AB93] [BC94] is an advanced constraint programming system based on Prolog. In recent years, it has been increasingly used for developing applications based on constraint solving methods [Ber90] [BDP94] [BCP92] [CDF94] [CF94].

For end-user applications, only a relatively small part of the total system will be concerned with problem solving. A large part of the development effort will be centred on interfaces to databases and other programs as well as the graphical interface for the user. In order to decrease the “time to market” of the finished product, the interface development should not take longer than the problem solver development.

Constraint applications are build mainly for the scheduling and planning area, as well as personnel planning and general assignment problems. Even if the problems solved are quite different, there are a number of common components which are used in nearly every system. A tabular textual output of data in a list or a graphical Gantt chart showing tasks and resources over time are typical examples of components which are needed in most constraint applications. In order to decrease development and maintenance costs it is necessary to reuse these components in all applications. Different applications may require application specific extensions of these components, so extensibility is a major design goal.

An important point both for maintenance and possible extensions is the use of a single, comprehensive data model throughout the application. This allows easy modification of the data at a single point of reference, which automatically updates the object model, data base interfaces and graphical components.

To obtain the maximum benefit from component reuse, we have developed an application framework which is now used for all new developments. This framework on one side is general enough to handle very different types of application problems, from oil refinery simulation to transportation problems in the food industry. On the other side it proposes a consistent interface style among applications, a common design of user interaction and of data management.

In this paper we describe this application framework and its components, and give some results on its use for large scale application development. The paper is organised as follows. We start with a brief overview of related work on component reuse and then describe some fundamental features of the CHIP system. In section 5, we describe the methodology which we use to generate application specific code. This discussion begins with the data model and then describes how we derive information about interfaces and graphics from it. In the section 7 we give an overview of a number of applications using this methodology. One of these applications, called TACT, will be presented in more detail. Section 8 contains an analysis of the benefits of our approach, giving details on the amount of reuse achieved from different applications. We will conclude with a discussion of the problems encountered during the development of the software components.

3. Related Work

In this paper we can not provide an overview of the current research in component reuse. An introduction to the literature can be found for example in [FI94]. The concept has become popular in recent years to overcome some of the perceived limitations of object oriented development [Ude94]. The main benefits obtained by reuse techniques are quoted as:

- decreased code size
- increased product quality

- increased development speed
- decreased maintenance cost

At the same time, a number of possible problems have been identified:

- difficulty of finding abstraction level
- more complex design
- increased start-up cost
- management complexity

We will discuss these limiting factors again in section 9. Systematic re-use of Prolog programs is mentioned in [OK90], but in general little experience with large-scale, end-user applications has been reported.

In order to speed-up development of decision support systems, a number of other approaches have been advocated:

1. Graphical interface builders can be used to quickly generate a skeleton of applications, defining interface elements like dialogs and windows. As these tools are not specialised to a particular application domain, they miss graphical facilities particular to decision support systems. In addition, these tools do not provide possibilities to define the non-graphical part of the application. Reuse of existing code is not simplified by these tools.
2. Modelling languages and model builders from Operations Research allow to define problems in terms of basic constraints. These tools can be useful to quickly define a problem solver based on specialised methods. They normally do not offer facilities to create user-friendly, interactive applications.
3. Application systems offer a possibility to define a problem completely within one framework without specialised coding. This approach is quite successful for narrowly defined application domains. A number of such systems using constraints have been developed. The basic limitation is that the expressive power of such tools must be restricted to a rather narrow field and that application specific extension can not be easily integrated.

Our approach tries to combine some of the benefits of the different methods. Based on a single data model, it simplifies the task of creating graphical interfaces and provides basic data manipulation facilities. At the same time, the constraint solver can be specialised to each particular problem with limited affect on the interface components.

4. CHIP

CHIP [DHS88] [VSD92] is a constraint logic programming language combining the declarative aspect of Prolog with the efficiency of constraint solving techniques. It extends conventional Prolog-like logic languages by introducing constraints over finite domains [AB93] [BC94], Boolean and

linear rational terms. CHIP has been successfully applied to a large number of industrial problems especially in the areas of planning [BDP94] [BCP92], manufacturing [CDF94], logistics [BKC94] [CF94], circuit design and financial planning [Ber90]. Originally designed at ECRC, CHIP is further developed and marketed by COSYTEC.

In this paper we do not discuss the different constraint solvers inside CHIP, but concentrate on other parts of the CHIP development environment. Figure 1 shows the different modules which are part of the CHIP solution environment.

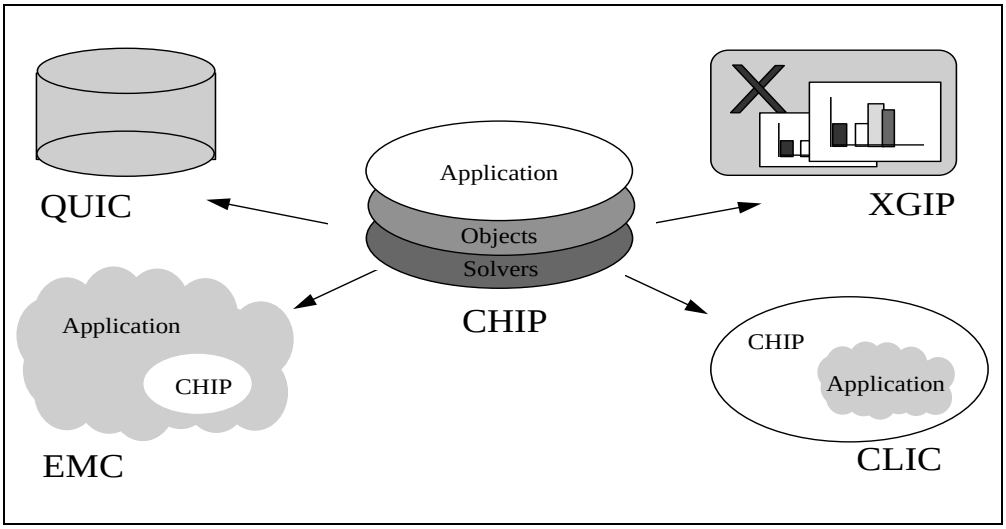

Figure 1: CHIP Modules

The *CHIP*++ object layer provides a feature-based [SA89] object system on top of the Prolog environment. CHIP++ offers basic object mechanisms like classes, instances and single inheritance. Methods are specified as Prolog clauses. The system provides full meta-level object control and introspection. This can be used for meta programming and program analysis purposes.

The *XGIP* graphical modules provides a high-level interface to the Xwindows system. Rather than simply mapping the existing API into Prolog, it contains more abstract primitives to manipulate windows and to perform drawing operations. A number of graphical libraries are provided on top of the XGIP system to handle panels and menus.

The *QUIC* module connects CHIP to relational database management systems via an SQL interface. SQL commands can be used inside CHIP to manipulate data stored in external database systems.

The CHIP system can be integrated with C code with the *CLIC* and *EMC* modules. These modules give the possibility to include C-code inside CHIP applications or to call CHIP from user-written C programs.

The CHIP constraint system is also available as a C/C++ library, which can be used as a standard component in other application. The application environment presented in this paper uses the Prolog based version of CHIP. The use of a very high level language can significantly decrease the development effort.

5. Methodology

We now start by describing our methodology for defining applications which is used as the basis for the reusable components. We begin with a discussion of the data model description underlying our applications. This data model provides a single point of reference for the different aspects of the data manipulation. This means that modifications and extensions of the model are localised within a single place. From this data model in Figure 3, a number of other designs are derived:

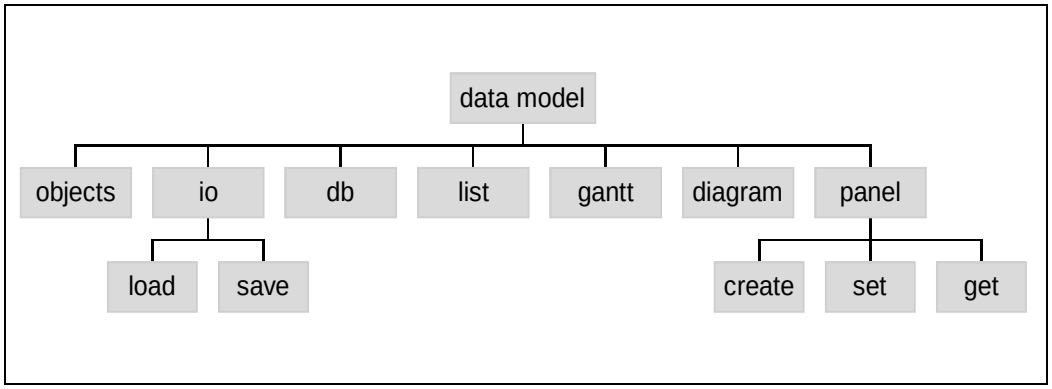

Figure 2: Description dependencies

The object model of the application is expressed in CHIP++ and corresponds to the data model, ignoring some interface aspects. The data model is also used to provide file based input/output used to locally store and recall application data. A data base model of the application is derived from the data model. Using meta programming, the necessary SQL code to create, delete, or access the data is automatically generated.

The data model is also used for defining aspects of the graphical interfaces. A list based manipulation of the data is possible as well as interfaces to Gantt charts and diagrams. With another meta-programming module, we automatically generate code for dialogs to interactively manipulate objects. We will now describe the different aspects in more detail.

5.1.1 Data Model

The data model of the application is used to describe the different objects which are defined inside the application. We can describe classes of objects based on other classes and their attributes. Attributes which are manipulated by the user can be defined with a number of different data types. Besides the primitive data types *integer*, *rational* and *atom* we offer *enumerated* types, time related types like *date* or *time*, graphical types like *colour* or *references* to other objects. We can give default values or range limits on the different data types, or restrict the size of a field. If necessary, we can also add graphical or interface constraints which are enforced in the graphical representation.

This data model provides a very rich data description, which can then be used to derive other aspects of the application. An important point is the centralised storage of different aspects of the data in a single place. This allows simple modification and extension of the data model.

5.2 Objects

The class structure of the application is obtained in a natural way from the data model. Each class is expressed as a CHIP++ class, with default values expressed as constraints or demons on the attributes. This object structure can grow to be quite complex. In Figure 3 we show an example from the FORWARD [Tec94] application. Classes are shown as rectangles, with inheritance displayed by straight lines and object references between classes are shown as arrows.

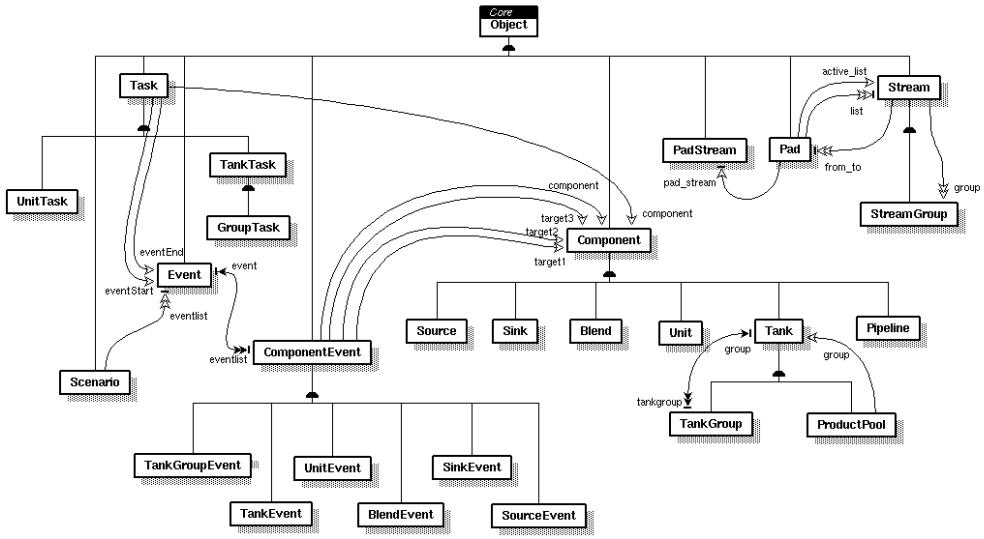

Figure 3: Class structure of application

5.3 Data Base connection

The data model can also be used to build a data base connection for the application. From the data description, we generate by a meta program the SQL DDL (Data Definition Language) format to create data base tables or views. Integrity constraints on the database are derived automatically from the data description or can be added manually as part of the data model. In a similar way we automatically generate the necessary QUIC code for the data base access. The meta programming facilities of Prolog in CHIP greatly simplify these tasks.

If the application uses an existing data base which is not consistent with the date model, the necessary interface routines must still be generated by hand.

```
%Class  Attr        Column_name       Label         Type    Width Edit Default Attrs Oracle Comment
d(crew, name,       crew_obj_name,    'Object',     atom,     10, n, -, [], ['NOT NULL'], '').
d(crew, class,      crew_class,       'Class',      atom,     10, n, -, [], ['NOT NULL'], '').
d(crew, id,         crew_id,          'Id',         atom,     10, n, 0, [], ['PRIMARY KEY'], 'Must be unique').
d(crew, number_crm, crew_number,      'Number',     integer, 10, y, 0, [], ['NOT NULL'], '').
d(crew, crew_name,  crew_name,        'Name',       atom,     10, y, -, [], ['NOT NULL'], '').
d(crew, first_name, crew_first_name, 'First Name', atom,     10, y, -, [], ['NOT NULL'], '').
d(crew, qual,       crew_qual,        'Qualif',     toggle,  10, y, -, [], ['NOT NULL'], '').
d(crew, rotation,   crew_rotation,    'Rotation',   atom,     10, y, -, [], ['NOT NULL'], '').
```

Figure 4: Data description

5.4 Interface Components

We now come to the description of different interface components which use the data model as a basis. These components are rather general and can be used in nearly every decision support application. In most cases, application specific behaviour must be added to link the components to the data.

5.4.1 List

The list object provides a textual view into a table of data of different formats. A typical example is shown in Figure 5. The list shows sets of objects of the same class. Each line corresponds to one data item, each column to one attribute of the class. Depending on the data model, different fields will use different display formats. The user can scroll horizontally and vertically in the list and interactively modify certain fields. Default values and constraints are used when we create new object instances or modify existing ones.

A set of messages is provided to change the list under program control or to query its status inside a program. User defined call-backs are executed when entries are modified. This allows the adaptation of the behaviour for particular applications.

Packaging Request	Finished Product	Qty rem	Qty prod	Reason	Line	Start	End
52	APRU613	14000	6000	IFM	DR	08 Mar 13:00	08 Mar 18:36
81	APAR4	8820	0	IFM	DR	11 Mar 08:00	11 Mar 11:31
33	APRU833	30000	0	IFM	DR	11 Mar 12:30	12 Mar 06:30
82	APRU840	15000	0	IFM	DR	12 Mar 06:30	12 Mar 15:30
80	APRU838	16000	0	IFM	DR	12 Mar 16:30	13 Mar 02:06
76	APM03	40000	0	IFM	DR	13 Mar 08:00	14 Mar 08:00
54	APAV102	5100	0	IFM	DR	14 Mar 21:00	15 Mar 00:03
49	APRU942	18600	0	IFM	DR	22 Mar 05:00	22 Mar 12:19

Figure 5: List example

5.4.2 Gantt

The Gantt chart is another of the high level interface components. It displays time in the x-axis and a set of resources in the y-axis. Tasks are shown on a background grid displaying the time resolution.

The drawing of the background, of resources and tasks is controlled by call-back predicates defined as part of the object description. User interaction like selection, movement or re-sizing can also be modified with call-back procedures. There are different alternatives for constrained or unconstrained movement. Depending on some attribute settings, a task may be moved in time and/or on resources. Limits on the movements are possible, as well as moving groups of tasks together.

Different messages can be used to modify the Gantt under program control Figure 6 shows a typical Gantt chart example.

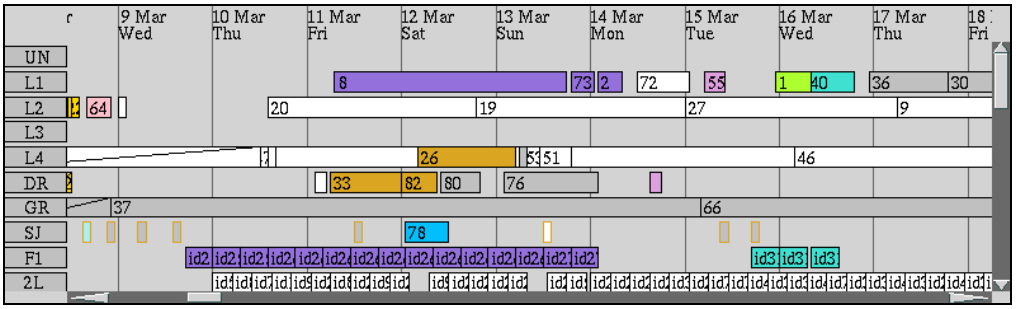

Figure 6: Gantt chart example

5.4.3 Diagram

Diagrams are used to show the evolution of values over time, for example stock levels, costs or cumulative resource requirements. Different forms of their graphical representation are possible with the diagram class object. Figure 7 shows a typical example. Two different types of curves, one solid, one outlined, are shown over a time scale and a value grid. These scales change as the user zooms in or out of the picture. At the top, a representation of events influencing the diagram levels is shown. They can be selected to obtain more information about the evolution of levels in time.

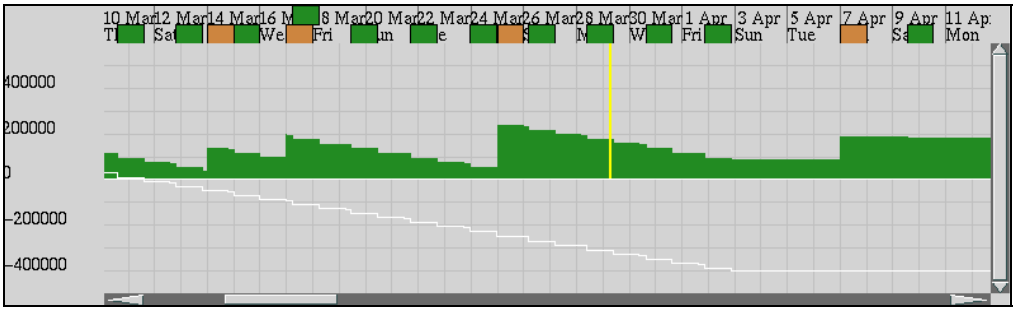

Figure 7: Diagram Example

5.4.4 Panels

For many purposes, the data manipulation with the list and the Gantt object will be sufficient for end-user manipulation. If required, the data model can also be used to build dialog windows in which attributes of one object can be modified as a set. The code to create the dialog panel and to set and exchange values between the object and the panel is obtained from the data model. User specific information for layout can be added in the data model.

6. Role of Solver

In the previous section we have presented some aspects of the data manipulation inside our application development. In this section we discuss how this approach interacts with the constraint solving capabilities of CHIP.

As constraints are mainly used to solve hard decision support problems, it is not possible to continuously update the solution as the user modifies the data. A run of the constraint solver must be started by the user, when all data manipulations are finished. The constraint program then converts the data into the correct constraints and tries to find a solution. If a solution is found, the system updates the data (tasks, resources, etc.) with the right information and the program returns to user control. While the program is running, the system will not allow data manipulation. This ensures data consistency between the data and the constraint model.

Once a solution is found, the user can manipulate the result, change data or parameters and possibly re-run the solver. It is possible to freeze parts of the existing solution so that they will not be changed by the solver. This approach has a number of advantages, since it leaves the user in total control of the system. If he wants to change part of the solution manually, this can be done even if constraints will be violated. The application can be run manually and tested before the constraint solver is finished. This fact is very useful during development, as it allows to develop the constraint solver and interfaces independently, decreasing the development time.

In most applications the solver can also be used in a check mode, where it is used to detect possible constraint violations in a manually modified solution.

The aim of the development is the generation of decision support tools, which work together with the user rather than replacing the user with a decision making tool.

7. Industrial applications

7.1 Overview

ATLAS [SC94] is a multi-user tool for detailed production scheduling in a chemical factory for semi-process and continuous production. The application is based on a constraint model using numerous constraints on machines and consumable resources. The constraint solver is embedded in a graphical environment using all described components together with a relational databases to exchange information with other information system tools.

FORWARD [Tec94] is a simulation and scheduling tool for oil refineries. In this application, CHIP is linked to a simulation tool written in FORTRAN which calculates results from a schedule obtained with the CHIP solver. The basic constraints express a non-linear optimisation problem over a continuous domain. The gantt and diagram components are used by this application.

TACT combines planning, scheduling and assignment aspects for the operational fleet management in the food industry. The system uses several interacting solvers to create a schedule for drivers, lorries and other resources in a transportation problem. The system is used both to create working schedules, re-scheduling and to handle 'what-if' scenarios.

PILOT [BKC94] is a decision support application to re-schedule cabin and flight crew for an airline operating multiple bases. The problem solver in CHIP builds cycle graphs to automatically assign crews, taking into account geographic, operational and working rules. The application makes use of the gantt and list components.

DAYSY is another decision support application for airlines dealing with Day-to-day Resources Management. The system reschedules airline flight activities in a real-time environment and resolve allocation of staff to operational flights. The system uses the description methology described for database definition, lists and multiple gantt charts.

CPLAN is a decision support tool for solving project management tasks. The solver uses, amongst others, the cumulative constraint to allocate individual resources to tasks, respecting overall capacities. The application

uses a drawing area object to define tasks and relationships between tasks, a gantt and list allows the user to display and manipulate the schedule.

APACHE [DS91] is a stand allocation package for international airports. The allocation is subject to numerous constraints: restrictive parking, customs status, preferences, etc. Manual and automatic scheduling is supported via lists, Gantts and database connection.

7.2 Example

TACT combines planning, scheduling and assignment aspects for the operational fleet management in the food industry. The systems replaces a manual process which had become exhaustive and uneconomic to manage.

The application developed using CHIP and the reusable components described above provides a decision support system integrated into existing data management systems. Amongst other benefits the system reduces the time taken to produce a schedule from hours to minutes, respects all constraints not always taken into account in the manual schedule, provides real-time rescheduling and hence can manage daily events, provides what-if capability and management reporting, provides a centralised database and greatly increases business efficiency.

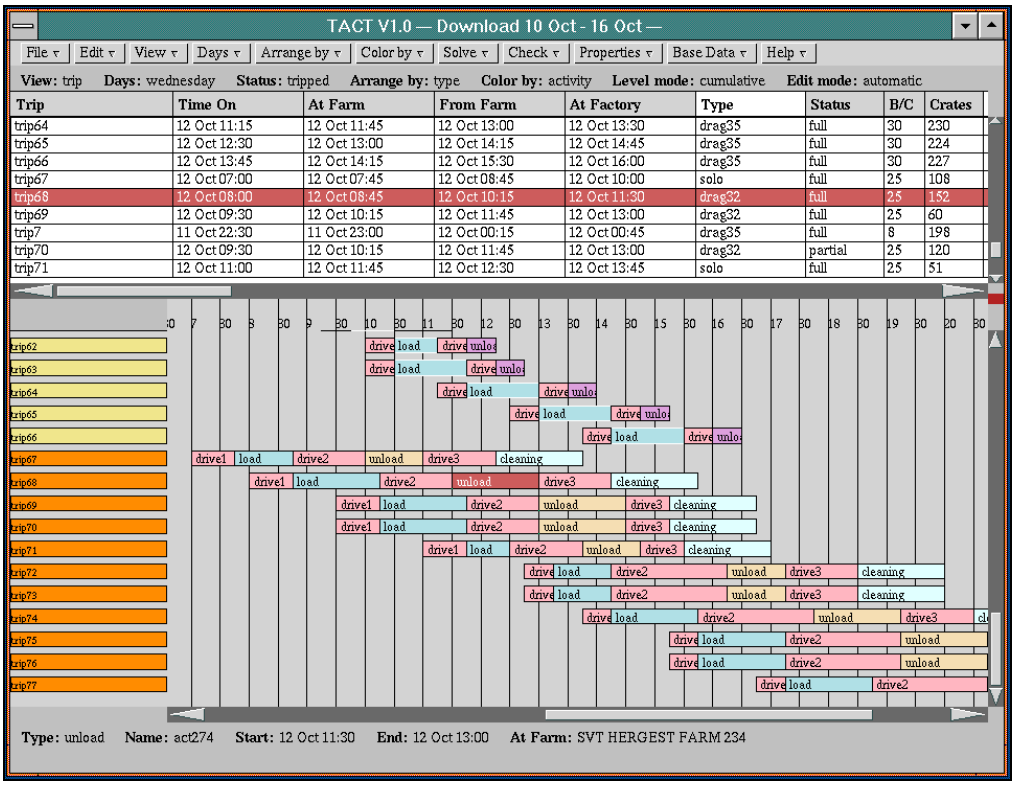

Figure 8: Application interface

Figure 8 shows part of the user interface developed. The list object provides a textual view on data elements for the application, these include the production and base data and the results of the assignment of activities. The user is able to scroll, select and edit fields in the list. A single list object is used to display all data and the user chooses the type of data to view via menu selection.

The Gantt chart provides a graphical representation of the textual information displayed in the list. Resources are displayed on the y axis and the allocation of tasks to these resources over time on the x axis. As an object is represented in both textual and graphical form, direct manipulation of either the Gantt or editing of the list results in updating of the object attributes and in turn graphical representation. A single Gantt object is used to display resources and tasks the user chooses the type of gantt view via pull down menus.

The application also displays stock levels in a diagram object and several transient panel windows for manipulation of application parameters. Through such panels it is possible to call and control the execution of the three solvers for the application. Manual scheduling is supported and tasks can be frozen, unscheduled, re-sized, moved, etc. The solver can be run in a passive manner to perform checks on the state of the assignment and produce warnings of violation of the constraint model.

8. Results

We now present some quantitative results on the reuse of components in our applications. Figure 9 shows the distribution of code parts in the ATLAS [SC94] application. This distribution is rather typical for programs with a data base connection and a finite domain solver, but without complex data creation inside the application. We find that the solver only accounts for 9% of overall code size. This is a clear indication for the efficiency of the constraint solving methodology compared to conventional design. Graphics routines and graphics libraries account for one third of the program. The static description part of the program is around 6 % of the overall code. Clearly, the majority of the program is concerned with database manipulations, reports and other integration parts. In this case, this includes a rather large amount of SQL and report writer code.

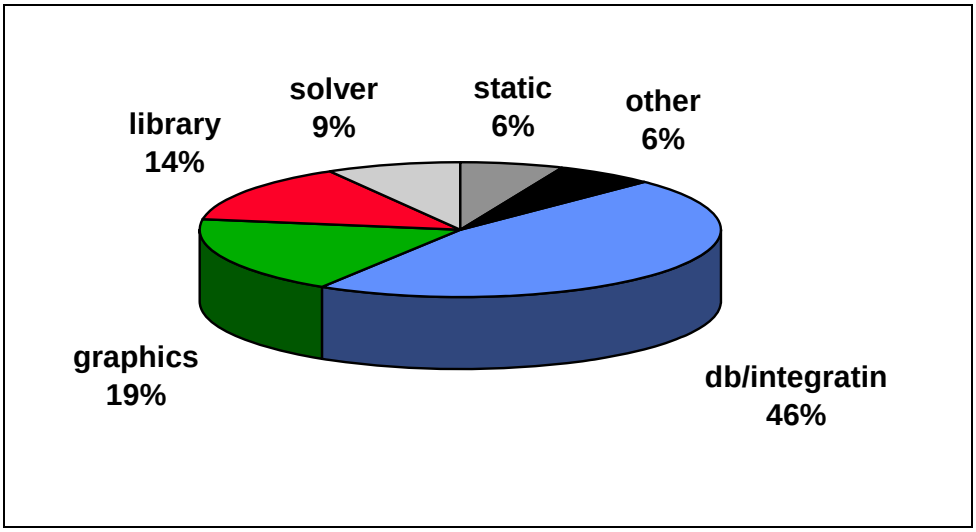

Figure 9: Code percentages

Our approach to reusable components has two effects. We can either reuse some application parts directly (like the Gantt or list objects) or we can generate code automatically from the data model. In one application currently under development, we have found the following proportions:

Part	Percentage
Generated Code	12%
Object libraries	18%
Reusable framework	20%

This means that around 50 % of the overall application were reused from existing code. The constraint model was developed completely from scratch. Some details of the interactive behaviour are also particular to this application. This shows that significant reductions can be achieved by reusing existing application components.

We can also note that the application framework is very useful during the design phase of a project. It is possible to create mock-ups and prototypes with limited functional behaviour quickly from the existing components. Many conceptual errors can be avoided by exploring different alternatives early in the design process.

9. Discussion

We now want to discuss some of the problems we encountered during the development process. A number of them are quite common problems with software re-use [FI94]:

1. The current version of the components has been refined over a period of several years and through several projects. It is very hard and expensive to define the correct level of abstraction beforehand or during a single

project. A significant start-up cost is required to make the components general enough to be easily reusable.
2. As component behaviour evolves, it becomes more and more difficult to keep existing applications current with the tool set. New functionality will not be used in older applications.
3. Good communications between the different development teams is essential for a successful reuse in a new project.

Some other problems are specific to our situation:

1. It is very hard to define and use complex data structures in pure Prolog. The structuring mechanisms provided in the object layer of CHIP++ offer a much better description format.
2. To obtain high-quality solutions we have to adapt the constraint model very closely to the particular problem. This reduces the amount of reuse which can be obtained for the constraint part of the application. Since the global constraints [AB93] [BC94] in CHIP already provide a very efficient modelling tool, this does not cause too much of a problem for rapid development.

While these problems demonstrate the difficulty of reusing large parts of applications, our experience is quite positive. Combined with the constraint solving capabilities of CHIP we now routinely use this framework for developing complex decision support systems. It provides one tool for the tasks of design, prototyping and production development of applications. Contrary to popular opinion, it is clear that Prolog based languages can be used to develop end-user applications with complex graphical interfaces.

10. Summary

In this paper we have presented a methodology for reusing CHIP components in the design of industrial applications. A single data model is used for the object description, database connection and graphical interface design. The major graphical components are a textual list, Gantt chart and level diagrams which appear in many constraint based applications. We have shown the systematic reuse of these components in a number of applications from quite different problem domains. Typical reuse rates reach around 50 % of the overall application size. This reuse both speeds up development and increases quality, as more already proven parts are integrated into the application. The use of a very high level language like CHIP with its object extension CHIP++ is very helpful, and in particular the meta programming facilities of Prolog.

11. References

[AB93] A. Aggoun, N. Beldiceanu
Extending CHIP in Order to Solve Complex Scheduling Problems
Journal of Mathematical and Computer Modelling, Vol. 17, No. 7, pages 57-73
Pergamon Press, 1993

[BKC94] G. Baues, P. Kay, P. Charlier
Constraint Based Resource Allocation for Airline Crew Management
ATTIS 94, Paris, April 1994

[BC94] N. Beldiceanu, E. Contejean
Introducing Global Constraints in CHIP
Journal of Mathematical and Computer Modelling, Vol 20, No 12, pp 97-123, 1994

[BCP92] J. Bellone, A. Chamard, C. Pradelles
PLANE -An Evolutive Planning System for Aircraft Production.
First International Conference on the Practical Application of Prolog. 1-3 April 1992, London.

[BC93] F. Benhamou, A. Colmerauer (Editors)
Constraint Logic Programming: Selected Research
MIT Press, 1993

[Ber90] F. Berthier
Solving Financial Decision Problems with CHIP
Proc 2nd Conf Economics and AI, Paris 223-238, June 1990

[BDP94] P. Bouzimault, Y. Delon, L. Peridy
Planning Exams Using Constraint Logic Programming
2nd Conf Practical Applications of Prolog, London, April 1994

[CDF94] A. Chamard, F. Deces, A. Fischler
A Workshop Scheduler System written in CHIP
2nd Conf Practical Applications of Prolog, London, April 1994

[CF94] C. Chiopris, M. Fabris
Optimal Management of a Large Computer Network with CHIP
2nd Conf Practical Applications of Prolog, London, April 1994

[DHS88] M. Dincbas, P. Van Hentenryck, H. Simonis, A. Aggoun, T. Graf and F. Berthier.
The Constraint Logic Programming Language CHIP.
In Proceedings of the International Conference on Fifth Generation Computer Systems (FGCS'88), pages 693-702, Tokyo, 1988.

[DS91] M. Dincbas, H. Simonis
APACHE - A Constraint Based, Automated Stand Allocation System
Proc. of Advanced Software Technology in Air Transport (ASTAIR'91)
Royal Aeronautical Society, London, UK, 23-24 October 1991, pages 267-282

[FI94] W. Frakes, S. Isoda
Success Factors of Systematic Reuse
IEEE Software, September 1994

[FHK92] T. Fruewirth, A. Herold, V. Kuchenhoff, T. Le Provost, P. Lim, M. Wallace
Constraint Logic Programming - An Informal Introduction
In Logic Programming in Action LNCS 636, 3-35, 1992

[JM94] J. Jaffar M. Maher
Constraint Logic Programming: A Survey
Journal of Logic Programming, 19/20:503-581, 1994

[OK90] R. O'Keefe
The Craft of Prolog
MIT Press, Boston, Ma, 1990

[SC94] H. Simonis, T. Cornelissens
Modelling Producer/Consumer Constraints
ILPS Post Conference Workshop on Constraint Languages/Systems and their Use in Problem Modelling, Ithaca, NY, Nov 1994

[SA91] G. Smolka, H. Ait-Kaci
Inherience Hierarchies: Semantics and Unification
Journal of Symbolic Computation, 7, 1989, pp 343-370

[Tec94] Technip
Forward Reference Manual
September 1994

[Ude94] J. Udell
Component Ware - Cover Story
Byte May, 1994

[VSD92] P. Van Hentenryck, H. Simonis, M. Dincbas
Constraint Satisfaction using Constraint Logic Programming
Journal of Artificial Intelligence, Vol.58, No.1-3, pp.113-161, USA, 1992

[Wal94] M. Wallace
Applying Constraints for Scheduling
In B. Mayoh, E. Tyugu, J. Penjaam (Eds) Constraint Programming, Springer Verlag, 1994