URC 97003

Issues in Defining Software Architectures in a GIS Environment,

Jesus Acosta and Lori Alvarado
Department of Computer Science
The University of Texas at El Paso
El Paso, TX 79968
{jacosta,alvarado}@cs.utep.edu

1 Introduction

The primary mission of the Pan-American Center for Earth and Environmental Studies (PACES) is to advance the research areas that are relevant to NASA's Mission to Planet Earth program [1], One of the act ivities at PACES is the est abolishment of a repository for geographical, geological and environmental information that covers various regions of Mexico and the southwest region of the U.S. and that is acquired from NASA and other sources through remote sensing, ground studies or paper-based maps. The center will be providing access of this information to other government entities in the U.S. and Mexico, and research groups from universities, national laboratories and industry.

Geographical Information Systems (GIS) provide the means to manage, manipulate, analyze and display geographically referenced information that will be managed by PACES. Excellent off-the-shelf software exists for a complete GIS as well as software for storing and managing spatial databases, processing images, networking and viewing maps wit, h layered information. This allows the user flexibility in combining systems to create a G IS or to mix these software packages with custom-built application programs. Software architectural languages provide the ability to specify the computational components and interactions among these components, an important topic in the domain of GIS because of the need to integrate numerous software packages.

'This paper discusses the characteristics that architectural languages address with respect to the issues relating to the data that must be communicated between software systems and components when systems interact. The paper presents a background on GIS in section 2. Section 3 gives an overview of software architecture and architectural languages. Section 4 suggests issues that may be of concern when defining the software architecture of a GIS. The last section discusses the future research effort and finishes with a summary,

2 Background

A Geographical Information System (GIS) is a computer-based system capable of assembling, storing, manipulating and displaying geographically referenced information, i.e., data identified according to its location. GIS technology can be used for scientific investigations, resource management, and development planning all within the scope of PACES. The types of analysis that this system must provide are measurement, mapping, monitoring, and modeling usually called the four M's of a GIS [7]. *Measurement* is concerned with storing data with various environmental parameters. A GIS must be able to retrieve old data and collect new data both efficiently and accurately.

The information may be combined with data from different sources to provide data about a region. A concern, then, is the ability to form comparisons on separate types of data. The feature of GIS that deals wit h integrating different types of data is referred to as *mapping.* For example, a system may allow one to relate information about the amount of rainfall to an aerial photograph of a region, providing information about which areas of the region tend to dry Up at certain times of the year. Mapping data from numerous data sets permits more intense studies of regions to be conducted.

A GIS should also allow a user to *monitor* changes in data over periods of time. For example, if a GIS can store data measurements taken at different points in time, a researcher can monitor changes that may occur in these areas.

Modeling allows a user to create models based on relations, For example, given an aerial photograph of a region and elevation readings taken from remote sensing, a model of a region could be produced allowing a more detailed study of the landscape.

Systems that provide the features discussed here can be created using off-the-shelf software. Software packages such as ESRI's ArcInfo[1], exist that provide all of the features in one package. ArcInfo is a series of six integrated software modules that provides basic GIS tools and utilities for cartographic design, query data entry and editing, data translation, polygon overlay and buffering (image processing), network analysis, and modeling. A GIS can also be constructed by combining the different, software packages that provide features necessary in such a system. A database management system such as Oracle7 can be used to store, manage and query spatial or multidimensional data. Image processing software, such as PCI's ACE[2], provides a complete environment for producing digital maps, integrating both vector and raster information. In addition, it includes a complete cartographic editing environment. Networking software such as Novell could also be used to create a GIS. An issue that is addressed in a later section is the need to specify the interface of these packages in order to provide the necessary information for successfully integrating software in this environment.

3 Software Architecture

Software architecture, an area in software engineering that has emerged seriously in the last decade, deals with the ability to formally describe the behavior of software systems. The software architecture of a system is defined in terms of computational components and the interactions among those components. When constructing a system, software architecture defines the different functionalities of the components that make up the system. This can be used to analyze the system and to determine whether the system will function properly. When integrating software systems, the software architecture is imperative when determining if the system accomplishes what is intended,

The languages that, are used to specify the architecture of software systems are called Architectural Description Languages or ADL's [4]. In order for an AD], to be effective, three common characteristics are traditionally addressed. The first of these is the components or modules of a system. These are the parts of a system that perform actions or functions. Components are such things as clients and servers, databases, filters, and layers in a hierarchical system [5]. The second characteristic is the connections or interactions between the specified components. Interactions among components at this level of design can be things such as procedure calls and shared variable access as well as any data that is passed between components of the system. The final characteristic that is addressed in an ADL are constraints. Constraints are usually expressed on the interface of a component and arc used to define restrictions.

Experimental languages for defining architectures include Rapide, MetaH and LILLEANA [8]. One of the architectural specific. at, ion language that we are examining is Rapide. A component in Rapide is specified through its interface, behavior, and constraints [3]. The interface identifies the types of events that a component can receive and generate by declaring in actions and out actions. The behavior section contains type declarations, objects, and a set of transition rules that operate on those objects and types. Objects and types model the state of the component, and transition rules model how objects and types react to actions received or generated by the component. The constraint section of a component constrains behavior by defining patterns of execution.

Consider an alarm system, as shown in Figure 1. The components of the system consist of an *Alarm Control* component, two light components and a speaker component. The light components, marked as *Ready Light* and *Armed Light*, simply display light based on input signals. The two possible input signals to these components are *off* and *on*. The speaker component is similar in function to the light components. The only difference is that the speaker component sounds an alarm instead of displaying light. The *Alarm Control* component controls the entire alarm system. The *Alarm Control* provides two kinds of external input actions: readings taken in through a sensor and codes taken in through a key pad.

An example of the AlarmControl component defined in Rapide is given below:

```
< global types >
      type signal is ...;
      on : signal := . . .;
```

[1] See http: //www.innovsvs. mm/.
[2] See http://www.pci.on.ca/.

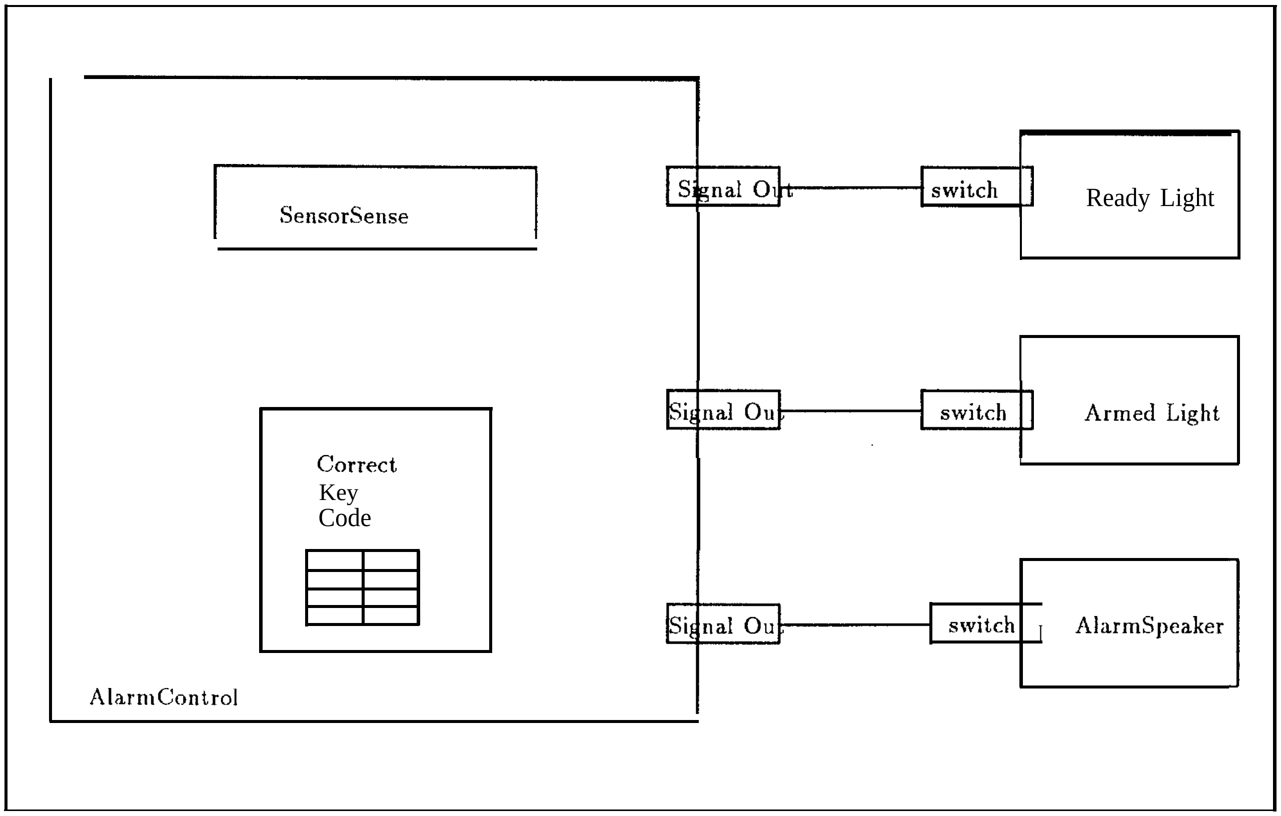

Figure 1: An alarm control component.

```
        off : signal := . . .;
interface AlarmControl is
        in action Correct. Key_Code;
        in action SensorSense;
        out action Alarm (S:signal);
        out action Armed(A:signal);
        out action Ready(R:signal);
behavior
        type state_type is (armed, ready, activated)
        state: state-type := ready;
        Correct_Key_Code where (state = ready) → state := armed;
                Armed(on)|| Ready (off);;
        Correct. Key_Code where (state = armed) → state := ready;
                Armed(off)|| Ready (on);;
        Correct-Key-Code where (state = activated) → state := ready;
        Alarm(off);;
        SensorSense where (state = armed) → state := activated;
        Alarm(on);;
constraint
        (
        [ Correct-Key-Code where (state = ready) →
                (Armed(on)||Ready(off)) ] <
        [ SensorSense where (state = armed) →
                (Alarm(on)) ]
        )
end Alarm Control;
```

The Rapide Language describes a component in terms of its interface, behavior within the component, and constraints on transitions that occur based on actions. [n the specification above, **in actions** arc actions that can be received from outside of the **AlarmControl** component. These include **Correct_Ke y_Code** and **SensorSense**. Actions generated with data passed out, of a component are specified as out **actions.**

Within the behavior block. actions that are generated as well as state transitions based on actions are specified. For example, if a Correct _Key_Code action is received, and the state of the system is armed, the transition from armed to ready occurs. The out actions Armed and Ready pass parameters of type **signal** out of the component. The constraint imposes an order on the actions using the < sign. The constraint above is stating that the alarm must be armed before it, can be activated.

4 Issues

There are a number of issues that are of concern when addressing systems in the GIS domain. These issues involve storage of datasets, handling of multiple formats, precision issues, and the integrity of the data in a GIS. GIS systems typically handle large amounts of data that are stored in one or more databases. When discussing software architectures in this domain, specifying how data is stored, retrieved and analyzed are central issues.

A GIS typically handles many different types of file formats, such as raster, vector, and image data. This is also a characteristic that is not, present in traditional systems. For example, if a developer wishes to use an existing off-the-shelf database to store much of the data, and an image processing software package that provides mapping features, in conjunction with a custom-built application, determining how the multiple packages store and operate on data must be communicated to the relevant stakeholders. Some of the questions that should be asked include the following [2]:

1. How are the numbers stored?
2. How is the data organized?
3. What is the dimensionality of tile data?
4. Is the data on a grid?
5. What is the best way to analyze the data?

It should be noted that numerous groups have been organized to address standards on data, for instance, compressing, holding and transmitting images. Example standards in this area include CompuServe Graphics Interchange Format (GIF) and Encapsulated PostScript Format (EPS). Standards also exist for spatial data exchange for transferring digital cartographic data anti vector/raster spatial data, e.g. the Spatial Data Transfer Specification (SDTS), US Digital Cartographic Data Standards Task Force (DCDSTF). In addition, there are standards set for database management systems [7].

An issue in using off-the-shelf packages relates to the difficulty in determining behavior of the software due to the lack of documentation; manufacturers typically do not provide documents that describe the structure of their software. Information provided in user's manuals do not address internal behavior of a system. How a package interfaces with other packages, however, can be determined through the documentation that is provided, as well as through analysis. Our focus is on defining the interfaces of packages used in GIS, and to determine if the available languages are effective architectural specification languages.

Another issue that must be considered in a GIS is the precision of data. Data used for geographical studies can vary greatly in precision. Because higher resolution of data requires considerably more storage and processing time, multiple data sets may be available for the same physical area with varying resolutions. A developer who plans to interface off-the-shelf software with a custom-built application must know how the data will vary in precision. For example, if a database package uses 32 bit numbers, and an image processing tool that will interface with the database uses 64 bit numbers, there may be a conflict which will cause loss of data when analysis occurs. Providing specifications of interfaces of the software packages that include this inform at ion would alert a developer to these types of changes.

The final issue deals with the support, of integrity checking. In a geographical information system, data integrity is critical as data is transferred from one software component to another. Invalid data input can significantly modify information used in a study. It is essential that architectures, and the languages used in the GIS arena address all these issues.

5 Future Work

The traditional view of software architectures and architectural description languages are designed to address general systems. in the domain of geographical information systems, issues exist which may or may not be addressed by the traditional view of architectures or existing architectural specification languages. The focus of the ongoing research is in further identifying design issues present in the GIS domain and determining what is necessary to describe a system architecturally. It also deals with being able to use a language to describe an off-the-shelf package in terms only of its interface.

It is **not known** whether this can be done effectively **with** existing languages. Part of the work involves examining the effectiveness of existing languages in specifying geographical information systems. Future interests include developing a survey instrument that can be distributed among software engineers and stakeholders in the GIS field. This instrument is needed to refine the issues explained above, as well as to identify additional issues that may need to be addressed in software architectures of geographical information systems.

Acknowledgments. This work was partially supported by NASA contract NCCW-0089 and NSF grant no. CDA-9522207.

References

[1] University of Texas at El Paso, "Proposal for the Pan American Center for Earth and Environmental Studies," submitted to NASA Headquarters, March 1995.

[2] Fortner, B., —-em The Data Handbook A Guide to Understanding the Organization and Visualization of Technical Data. Santa Clara, CA: TELOS, 1995.

[3] Luckham, D., et al. "Specification and Analysis of System Architecture using Rapide", IEEE Transactions on Software Engineering, 21(4):336-355, April 1995.

[4] Luckham, " Micro Rapide: An Architecture Description Language", Journal of Systems and Software, 21(3):253-265, June 1993.

[5] Shaw, M., Garlan, D., "Software Architecture", Toronto: Prentice Hall, 1996.

[6] Rapide Design Team, " Rapide 1.0 Syntax Summary", Program Analysis and Verification Group, Comput. Syst. Lab., Stanford Univ., version 1. cd., August 1993.

[7] Worboys, M., "GIS: A Computing Perspective", Bristol: Taylor and Francis, 1995.

[8] Vestal, S., " A Cursory Overview and Comparison of Four Architecture Description Languages", Honeywell Technology Center: 18 Feb. 1993.

[9] Wiederhold, G., Wegner, S., "Toward Megaprogramming", *Communications of the A CM, 35(11): 89-99, Nov. 92.*